Department of Mechanical Engineering
University of Canterbury

Masters Thesis

POSE ESTIMATION FOR FEEDBACK CONTROL IN A SNAKE ROBOT

Author:
Callum Scott

Supervisor:
XiaoQi CHEN

31 December 2017

Abstract

This thesis presents a pose estimation algorithm for a snake robot as a step towards creating a closed-loop controlled snake robot. The University of Canterbury snake robot is being developed for use in urban search and rescue, where snake robots have the advantage over wheeled robots that they are able to navigate through complex environments. For the snake robot to be able to make intelligent control decisions, feedback is required from the snake robot using sensors. Sensor data from an IMU and motor encoder is collected from each module through a data bus from a base station connected to a PC. The PC uses the Robot Operating System framework to run software for the pose estimation algorithm. The sensor data is collected through the serial node running on the base station, which is then sent to the pose estimation node running the pose estimation algorithm.

The pose estimation algorithm predicts the pose of each module of the snake robot and the orientation of the snake robot as a whole. A square root spherically simplex unscented Kalman filter was used due to the highly non-linear measurement model used. A virtual chassis was used to abstract away from the internal shape of the robot to allow it to be treated as though it were a wheeled robot. The linear progression, turning, rolling and rotating gaits were all used to test the pose estimation algorithm with a gait based model used for each. An additional joint angle model which does not require prior knowledge of the gait being performed was created to compare the two different approaches. The motion gaits were tested using the Vicon motion tracking system to compare the predicted angle of each module to the ground truth from the Vicon system.

The inchworm gait using the gait parameters model had a mean roll error of 1.65 degrees and a mean pitch error of 1.82 degrees compared to the joint angles model with a mean roll error of 1.84 degrees and a mean pitch error of 2.12 degrees. The rotating gait using the gait parameters model had a mean roll error of 2.34

degrees and a mean pitch error of 2.59 degrees compared to the joint angles model with a mean roll error of 2.58 degrees and a mean pitch error of 2.51 degrees. The yaw prediction was inaccurate due to being based on a magnetometer located in the head module which was adversely affected by magnetic interference from the building.

The pose estimation algorithm was designed to be redundant so that if a module fails, the algorithm is still accurate. To test this, the sensors on module 4 were set to not return sensor data. The mean roll and pitch errors for module 4 using the rotating gait with the gait parameters model were 3.02 degrees and 2.62 degrees respectively, only slightly worse than with no sensor failure. To simulate a hardware failure, the motor in module 4 was set to a constant 0 degrees while using the rotating gait. The mean prediction error of module 4 was slightly higher with a mean roll error of 4.42 degrees and a mean pitch error of 4.33 degrees. The head and tail modules were similarly affected by the hardware failure with slightly increased mean errors.

It was found that the accuracy of the joint angle model is very similar to the gait parameters model. As the joint angle model can perform all of the motion gaits tested with the same process model, since it is not reliant on prior knowledge of which gait pattern is being used, it is the more practical choice of model as future research is conducted.

Deputy Vice-Chancellor's Office
Postgraduate Office

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

Co-Authorship Form

This form is to accompany the submission of any thesis that contains research reported in co-authored work that has been published, accepted for publication, or submitted for publication. A copy of this form should be included for each co-authored work that is included in the thesis. Completed forms should be included at the front (after the thesis abstract) of each copy of the thesis submitted for examination and library deposit.

Publication 1 (in print):

M.J. Koopaee, C. Gilani, C. Scott, X. Chen, "Bio Inspired Snake Robots: Design, Modelling and Control," in *Handbook of Research on Biomimetics and Biomedical Robotics*, Cairo: IGI Global, 2018.

Components of thesis extracted from Publication 1:

Chapter 3, particularly section 3.2
Chapter 6, particularly section 6.1

Publication 1 (in print):

The contents of the chapter were supervised by Prof. XiaoQi and first-authored by Mohammadali Javaheri Koopaee. The student contributed particularly on the electrical and software design of a snake robot. Other co-authors prepared other contents of the chapter.

Certification by Co-authors:

If there is more than one co-author then a single co-author can sign on behalf of all
The undersigned certifys that:

- The above statement correctly reflects the nature and extent of the PhD candidate's contribution to this co-authored work
- In cases where the candidate was the lead author of the co-authored work he or she wrote the text

Name: *XiaoQi Chen* Signature:  Date: *22/12/2017*

Acknowledgements

I would first like to thank my supervisor Prof. XiaoQi Chen for his guidance of the project. I would also like to thank my co-supervisor Dr. Chris Pretty for his help in preparing my thesis.

A special thanks goes to Julian Murphy for ordering all of the equipment I needed, creating the PCBs and offering technical advice on the hardware of the snake robot.

I would also like to thank all of the visiting students who contributed directly and indirectly to the snake robot project to push it forward. A final thanks goes to all the postgraduate students in my office for making my Masters project a great experience.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Objectives 2

1.3 Thesis Outline 3

2 Literature Review and Project Background **5**

2.1 Biological Snakes 5

2.2 Snake Robots 8

2.2.1 Early Snake Robots 8

2.2.2 Recent Snake Robots 9

2.3 University of Canterbury Snake Robot 11

2.3.1 Snake Robot Design 12

2.3.2 Control Design 13

2.3.3 Software Design 15

2.4 Kalman Filters 16

2.5 Other Institutions Feedback Systems 19

2.6 Summary 21

3 Hardware Design for Sensor Data Collection **23**

3.1 IMU Selection 24

3.2 Data Bus Selection for Robot Tether 25

3.3 Microcontroller Selection 28
3.4 Module Redesign 30
3.5 Base Station Design 30
3.6 Summary 31

4 Synchronising Sensor Data **32**

4.1 Solutions to Synchronise the Sensors 34
4.2 Sensor Synchronisation Implementation 37
4.2.1 Motor Profile 39
4.3 Summary 40

5 Pose Estimation Algorithm Design **42**

5.1 Virtual Chassis 43
5.2 The Unscented Kalman Filter 45
5.2.1 State Vectors 47
5.2.2 Process Model 50
5.2.3 Measurement Vector 51
5.2.4 Measurement Model 51
5.2.5 Difficulties Using Quaternions in the SR-SSUKF 52
5.2.6 Tuning the Kalman Filter 53
5.2.7 Failed Sensor Readings 54
5.3 Internal Pose Estimation Results 54
5.4 Summary 60

6 Software Design of Pose Estimation Algorithm **62**

6.1 The Robot Operating System 62
6.2 Gait Generation Node 64
6.3 Base Station Software Design 66

6.4 Snake Robot Module Software Design 69
6.5 Pose Estimation Software 73
6.6 Summary 75

7 Results and Verification **77**
7.1 Verification Setup 77
7.2 Verification Results 79
7.2.1 Motion Gait Tests 79
7.2.2 Further Tests 85
7.2.3 Module Failure 87
7.3 Summary 91

8 Conclusions and Future Work **93**
8.1 Conclusions 93
8.2 Future Work 96
8.2.1 Hardware and Software Design 96
8.2.2 Pose Estimation Algorithm Design 96

Bibliography **98**

Appendices **105**

A IMU Component Properties **106**

B Base Station PCB **108**

C UKF Scaling Factors **109**

D Internal Test Data **111**

List of Figures

2.1 Common gait patterns used for locomotion displayed by snakes showing the contact points with the ground. 6

2.2 The ACM-R3 developed by Hirose. 8

2.3 The ACM-R8 developed by Komura. 9

2.4 The Multi-Crawler active tread based snake-like robot. 9

2.5 Kulko snake robot developed by Liljebäck. 10

2.6 The Eelume robot developed for underwater applications in the oil and gas industry. 10

2.7 The Unified Snake Robot climbing a tree 11

2.8 The SEA snake robot. 12

2.9 The snake robot under development at the University of Canterbury. 13

2.10 The yaw and pitch modules of the university of Canterbury Snake Robot. 14

2.11 ROS communication network used by the snake robot. 16

2.12 The GUI created to control the snake robot. 17

2.13 Linearisation of a 1-Dimensional system using a UKF with a Gaussian distribution. 18

3.1 Data rate versus the cable length of the RS485 Standard. 26

3.2 Data rate compared to the cable length of the CAN Bus. 27

3.3 Schematic of the RS485 data bus connections between the Master and Slave devices and the required termination resistors. 28

3.4 Layout of the hardware to collect sensor data from the snake robot. 29

3.5 Layout of the hardware in a snake module. 30

4.1 The operation cycle of the snake robot software. 34

4.2 Motor actuation time compared to the sensor reading time. 35

4.3 Expected execution time of the sensor collection using the hardware solution. 36

4.4 Motor profile used by the Herkulex DRS-0101 Smart Servos. 36

4.5 New hardware design to collect sensor data from the snake robot. . 38

4.6 Comparison of yaw motor profiles at the head and tail of the snake robot with motor commands being sent every 100ms. 40

5.1 Comparison between the frame fixed to the head module and the virtual chassis. 44

5.2 Setup of the test area for the internal tests. 55

5.3 Motor angles of modules 1 and 2 while performing the rolling gait. . 55

5.4 Motor angle errors of modules 1 and 2 while performing the rolling gait. 56

5.5 Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the gait parameters model. 57

5.6 Predicted error of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the gait parameters model. 58

5.7 Comparison between the SR-SSUKF prediction using the gait parameters model and the joint angle model while using the rotating gait. 59

6.1 ROS graph of the nodes interactions where the ellipses are the nodes, the rectangles are the topics and the arrows are the subscriptions to those topics. 63

6.2 Transfer of data between the computer and the snake robot. 64

6.3 Callback functions used by the gait generation ROS node. 65

6.4 Callback functions used by the ROS serial node on the Teensy 3.2. . 68

6.5 Callback functions used by the Teensy LC task scheduler. 70

6.6 Priority of tasks used by the Teensy LC task scheduler. 71

6.7 Callback functions used by the ROS Snake UKF node. 74

7.1 Setup of the retroreflective markers used by the Vicon motion tracking system for the three modules. 78

7.2 The Vicon motion tracking software. 78

7.3 The pose estimation algorithm compared to the Vicon motion tracking system for the head module while performing the linear progression gait. 80

7.4 The predicted yaw of the head module using the linear progression gait. 81

7.5 The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the turning gait. 82

7.6 The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the rotating gait. 82

7.7 The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the linear progression gait with the joint angles model. 83

7.8 The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the rotating gait with the joint angles model. 84

7.9 The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the linear progression gait while moving down a 15 degrees ramp. 86

7.10 The pose estimation algorithm compared to the Vicon motion tracking system for the module 4 while performing the rotating gait where the sensor in module 4 has failed. 88

7.11 The pose estimation algorithm compared to the Vicon motion tracking system for the module 4 while performing the rotating gait where the motor in module 4 has failed. 89

7.12 The pose estimation algorithm compared to the Vicon motion tracking system for the head module while performing the rotating gait where the motor in module 4 has failed. 90

B.1 PCB of the base station. 108

B.2 Photo of the base station. 108

D.1 Predicted pose of modules 0 and 9 with the SR-SSUKF while using the linear progression gait with the gait parameters model. 112

D.2 Predicted error of modules 0 and 9 with the SR-SSUKF while using the linear progression gait with the gait parameters model. 113

D.3 Predicted pose of modules 0 and 9 with the SR-SSUKF while using the turning gait with the gait parameters model. 114

D.4 Predicted error of modules 0 and 9 with the SR-SSUKF while using the turning gait with the gait parameters model. 115

D.5 Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the gait parameters model. 116

D.6 Predicted error of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the gait parameters model. 117

D.7 Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rotating gait with the gait parameters model. 118

D.8 Predicted error of modules 0 and 9 with the SR-SSUKF while using the rotating gait with the gait parameters model. 119

D.9 Predicted pose of modules 0 and 9 with the SR-SSUKF while using the linear progression gait with the joint angles model. 120

D.10 Predicted error of modules 0 and 9 with the SR-SSUKF while using the linear progression gait with the joint angles model. 121

D.11 Predicted pose of modules 0 and 9 with the SR-SSUKF while using the turning gait with the joint angles model. 122

D.12 Predicted error of modules 0 and 9 with the SR-SSUKF while using the turning gait with the joint angles model. 123

D.13 Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the joint angles model. 124

D.14 Predicted error of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the joint angles model. 125

D.15 Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rotating gait with the joint angles model. 126

D.16 Predicted error of modules 0 and 9 with the SR-SSUKF while using the rotating gait with the joint angles model. 127

List of Tables

2.1 Optimal gait parameters found. 15

3.1 IMUs available using the researched components. 24

4.1 The commanded angle of all the motor angles compared to the Encoder angle. 32

4.2 The commanded angle of all the motor angles compared to the Encoder angle with the new hardware design. 38

5.1 Scaling factors for the linear progression gait. 53

5.2 Module 0 mean Euler angle prediction error. 60

6.1 Control flags for message structure. 67

6.2 Execution time of the callback functions performed by the Teensy 3.2. 68

6.3 Execution time of the five tasks performed by the Teensy LC on module 3. 73

6.4 Execution time of the Sensor Data callback function in the snake UKF node. 75

7.1 Vicon mean Euler angles error for the head module. 85

7.2 Vicon mean Euler angles error for the head module performing the linear progression gait while moving down a ramp. 86

7.3 Vicon mean Euler angles errors for the module failure tests. 91

A.1 Acceleration Properties. 106

A.2 Gyroscope Properties. 106

A.3 Magnetometer Properties. 107

C.1 Scaling factors for the rolling gait. 109

C.2 Scaling factors for the rotating gait. 109

C.3 Scaling factors for the joint angles model. 110

D.1 Internal test data mean error for modules 0 and 9. 111

Chapter 1

Introduction

1.1 Motivation

Mobile robotics have gained a lot of prevalence within industry for being able to efficiently reach places that people struggle to. Wheeled robots are the most common form of mobile robotics, which work well in predetermined environments where the terrain is known in advance. However in complex environments they commonly lack the necessary finesse and adaptability to navigate efficiently over or around obstacles. The development of snake robots is a useful avenue of research to augment mobile robotics which offers several advantages over more traditional wheeled robots in such environments:

- Their compact size allows them to fit through small gaps.
- They can navigate complex environments by using the terrain to propel themselves.
- Multiple modules provides a redundant design which allows the robot to continue operating even if modules fail.
- Snake robots use stable gait patterns which makes falling over hard. If the robot does fall over, the modular design means that the gait pattern can be adjusted and the robot can keep operating.
- Snake robot gait patterns can be used to operate in water.

A potential application for a snake robot is use in urban search and rescue after an earthquake. An earthquake causes a large amount of widespread damage creating a highly complex environment with collapsed buildings and rubble. Shortly after an earthquake is an important time where there is likely to be survivors trapped under rubble but is also the most dangerous time for rescuers due to unstable buildings. Mobile robots are used to assist the rescuers to find survivors without risking further loss of life. A snake robot would be able to navigate through the rubble and fit into small gaps to find survivors without risking further collapse of the buildings.

The snake robot could be used in a wide variety of industrial plant inspections in areas that are too dangerous or inaccessible for humans to enter such as around machinery or under floors. Snake robots lend themselves to operating in confined spaces such as these which allows for efficient safe inspections. The versatility of snake robots would allow them to perform a wide variety of tasks for industrial plant inspections.

The University of Canterbury is developing a snake robot for use in urban search and rescue or industrial plant inspections which uses open-loop control, meaning there is no feedback to the computer. For intelligent control of a snake robot, the operator should only be controlling the high level position of the snake robot like the direction and speed. The low level internal operation of the snake robot, such as the gait parameters that generate the motor angles of each module, should be autonomously controlled. For autonomous control, the robot should use a closed-loop control scheme which requires sensor feedback. To determine both the internal orientation of each of the modules and the orientation of the robot as a whole, a pose estimation algorithm is required.

1.2 Objectives

This thesis focuses on the development of a pose estimation algorithm as a step towards implementing autonomous control of the snake robot. The scope of the work is restricted to the gait patterns presented by Gilani who developed the mechanical design of the University of Canterbury Snake Robot [1]. The main objectives of the thesis are outline below.

Objective 1: Develop a pose estimation algorithm to determine the orientation of the snake robot in relation to a global reference frame.

The pose estimation algorithm inspired by previous studies should predict the orientation of the snake robot in a global reference frame. The design should be able to predict the orientation of the snake robot as a whole and the orientation of each individual module. Experimental validation will be performed on the pose estimation algorithm to determine its effectiveness.

Objective 2: Design the pose estimation algorithm to be fault-tolerant.

Due to the inherent nature of the snake robot having multiple devices on each module, it is difficult to completely eliminate sensor failure. The pose estimation should be able to detect failed sensor readings and ignore them in order to continue accurately predicting the pose of the snake robot. The snake robot uses a design redundancy so that the failure of one snake module does not prevent the robot from functioning. The pose estimation design should be able to predict the pose of a snake module even if a modules motor stops working.

Objective 3: Design a robust modular electrical and software framework to collect sensor data.

A modular approach should be taken for the development of the electrical and software design. Each module, excluding the head and tail modules, should contain the same electronics and run identical software so that adding or removing a module becomes trivial. Future designs of the snake robot are likely to contain additional sensors, so the software should be designed to allow for easy implementation of new hardware.

1.3 Thesis Outline

Chapter 2
Chapter 2 reviews previous literature of the development of snake robots and pose estimation techniques used. The background of the University of Canterbury Snake Robot is also explained to give the starting point for the work presented in this thesis.

Chapter 3
Chapter 3 presents the selection of hardware such as the sensors required to develop the pose estimation algorithm. The hardware layout of the snake robot is shown to explain how the sensor data is collected and returned to the PC.

Chapter 4
Chapter 4 discusses sensor synchronisation which was considered in the collection of sensor data. The motors were profiled to determine an issue caused by the power distribution system.

Chapter 5
Chapter 5 presents the developed pose estimation algorithm along with internal results of the algorithm tracking the collected sensor data.

Chapter 6
Chapter 6 describes the software framework used to create the pose estimation algorithm including the software on each of the modules to collect the sensor data.

Chapter 7
Chapter 7 presents the results of a comparison between the pose estimation algorithm and the ground truth from a Vicon motion tracking system.

Chapter 8
Chapter 8 presents the conclusions of the thesis as well as possible future work that could be undertaken.

Chapter 2

Literature Review and Project Background

This chapter provides a background to the snake robots that form the basis of this thesis. A brief history of snake robots starting with biological snakes is outlined, moving through early snake robots, followed by more recent snake robots of interest to the development of a pose estimation algorithm. The University of Canterbury snake robot is described to give an overview of the mechanical design that this thesis is based on. Finally, research into pose estimation algorithms and those used by other snake robots is presented to give an background of the algorithm presented in this thesis.

2.1 Biological Snakes

To understand the control design used in snake robots, it is useful to understand the biological counterparts that they are based on. Snakes are made up of multiple repeating sections, each providing a small range of motion in both the lateral and dorsal planes giving them the ability to utilise efficient locomotion gaits.[2]. Their unique vertebrae give them the ability to use these locomotion gaits to efficiently traverse complex environments.

Gray [3] in the 1940s suggested that there are four types of locomotion gaits used by snakes: lateral undulation (serpentine locomotion), sidewinding locomotion,

concertina locomotion, and linear progression (rectilinear locomotion), which can be seen in Figure 2.1

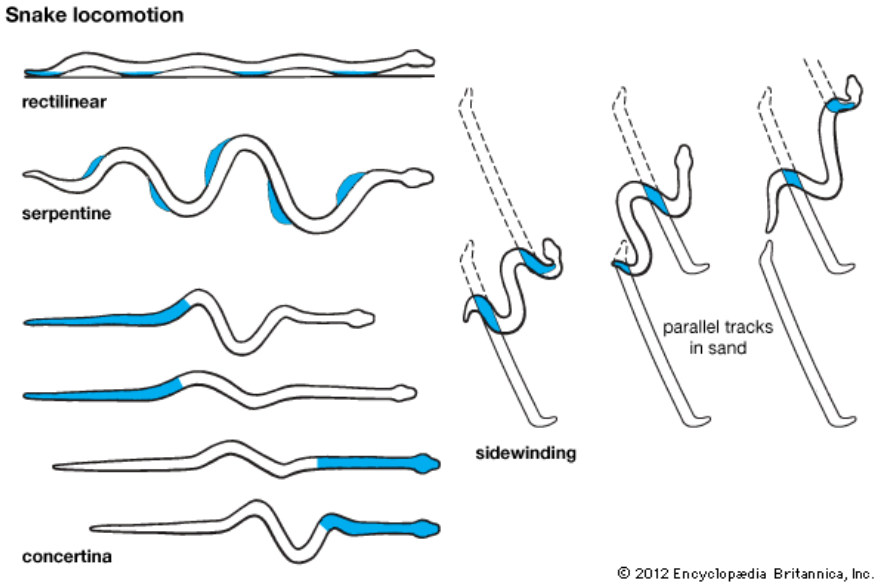


Figure 2.1: Common gait patterns used for locomotion displayed by snakes showing the contact points with the ground. (retrieved from [4])

Lateral undulation is an efficient gait with energy consumption equivalent to that of similarly sized legged animals, and is the most commonly used form of locomotion for snakes [5]. The gait pattern is generated by propagating a transverse wave through the snakes body such that all parts of the snake are moving at once. Every part of the snake follows the same path creating a track in the ground which helps to prevent lateral slipping. In order to achieve forward propulsion, the horizontal friction from the ground must be sufficient such that it is easier for the joints to move forward as opposed to sideways. Irregularities on the surface of the ground act as push points to give the required friction to enable forward propulsion [6]. Lateral undulation is not suited to situations where there is low friction or narrow passages such as pipes.

Sidewinding locomotion is an efficient locomotion gait for traversing loose, low shear surfaces, such as sand, and as such is most commonly observed being used by snakes in desert regions. The snake lifts sections of its body off the ground such that there are static contact points leaving short parallel lines at an inclined angle to generate forward propulsion [7].

Concertina locomotion is a less efficient gait often used in confined spaces such as tunnels where it is unable to use more efficient gaits. It is achieved by anchoring

a portion of the snake body and using it to push or pull the rest of the snake forward [8]. Forward locomotion is obtained when the friction from the anchored section of the snake is greater than the moving part.

Linear progression is the slowest gait which is more commonly used by larger snakes. The snake lifts sections of its body off the ground and places them further forward where they anchor them to the ground and pull the other sections to that point. Unlike the other described types of locomotion, rectilinear locomotion does not require lateral friction in order to propel the snake forward [8].

The choice of gait used by a snake is determined by the environment and the speed that it wants to go at. The efficiency of snake gaits patterns are at best as good as limbed animals but in many cases such as concertina locomotion and Rectilinear locomotion, are worse [5]. Limbless locomotion does however offer some advantages over limbed locomotion:

- By lifting up the front portion of their bodies, snakes are able to traverse obstacles significantly taller than their own body height.
- Locomotion gaits used by snakes are very stable.
- The large surface area of a snake provides good traction in complex environments.
- Multiple repeated sections of the snake allows redundancy such that an injury to one section does immobilise a snake.
- Snakes are versatile such that they are able to wrap their bodies around objects to grasp them.

The efficiency of snakes in a complex environment has inspired the creation of snake robots to replicate snake gait patterns.

2.2 Snake Robots

2.2.1 Early Snake Robots

Snakes are able to perform such a variety of different gaits that it is hard to design a snake-inspired robot able to replicate all of them. Most snake-inspired robots using wheels are designed to perform lateral undulation and are unable to use other gaits. Wheel-less snake robots are designed with a certain gait in mind such as lateral undulation which the mechanical design is optimised for but are often able to perform other movement gaits though less efficiently. Additional non snake-like gait patterns such as rolling can also be used by these snake robots.

Hirose's first robot, the Active Cord Mechanism (ACM) developed in 1972, used passive wheels [9]. He subsequently developed the ACM II and the ACM III. The ACM III used small wheels on the bottom of each link, and allowed a low friction coefficient in the forward direction and a high friction coefficient in the lateral direction, such that forward propulsion was achieved only through lateral undulation [10]. Hirose later improved the design with the ACM-R3 shown in Figure 2.2, which has large wheels on all sides of the robot. The new design is able to lift up its body to traverse low obstacles and allows the wheels to freely roll against contacted obstacles.

Figure 2.2: The ACM-R3 developed by Hirose. (retrieved from [2])

The ACM-R8 developed by Komura [11] is the most recent robot developed in the ACM series. The ACM-R8 is the successor to the ACM-R4.2 and is an active wheel snake robot. The primary goal of the ACM-R8 snake robot was to be able to climb steps taller than itself and have the potential to interact with the environment like

operating door knobs. Komura concluded that the robot was particularly good at climbing steps, and with a gripper attachment, would be able to operate door knobs. The robot lacked cameras which would need to be attached to improve the robots practicality.

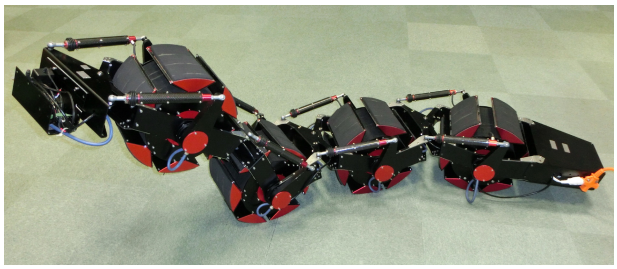


Figure 2.3: The ACM-R8 developed by Komura. (retrieved from [11])

2.2.2 Recent Snake Robots

A snake-like robot based on an active tread design was developed by Ito at Hosei University in 2016 [12]. The semi-autonomous, serially connected multi-crawler robot was developed for use in urban search and rescue operations. It is battery powered and controlled from a remote with the aim of being easy to use by untrained rescuers. Therefore the remote controls the macro operation while the robot passively controls the joints to avoid collisions. The robot has no sensors, instead the robot uses wires running along its length to constrain the joints such that the robot will deform around the obstacle and continue in the desired direction, which makes the robot easy to operate.

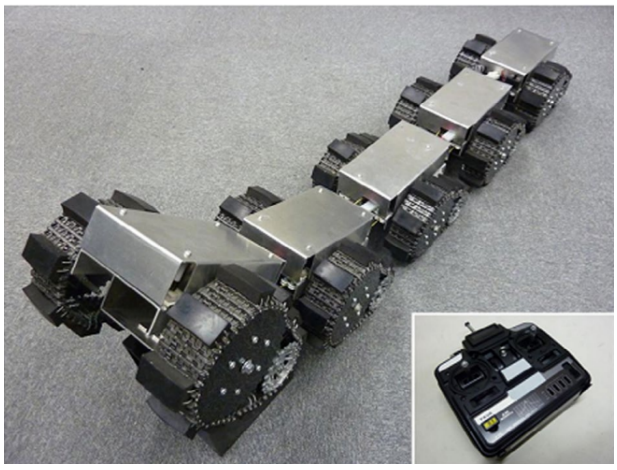


Figure 2.4: The Multi-Crawler active tread based snake-like robot. (retrieved from [12])

The Kulko snake robot was developed by Liljebäck in 2010 to improve the design

of obstacle aided snake robots [13]. He argues that to achieve intelligent obstacle-aided locomotion, the snake robot requires both a smooth exterior surface and a contact force sensing system. Kulko therefore uses sphere shaped modules, creating a smooth exterior surface, which houses the force sensing resistors used to detect obstacles. Though it is not accurate enough to make precise measurements, it is accurate enough to determine where the snake robot is in contact with terrain allowing the robot to make intelligent decisions to aid in its propulsion.

Figure 2.5: Kulko snake robot developed by Liljebäck. (retrieved from [13])

The Eelume is an underwater modular snake robot developed for commercial use in the oil and gas industry to perform inspections, maintenance and repairs. The Eelume uses snake gait patterns in conjunction with the thrusters to propel itself through the water [14].

Figure 2.6: The Eelume robot developed for underwater applications in the oil and gas industry. (retrieved from [15])

A modular snake robot was designed by Wright in 2007 at Carnegie Mellon University [16]. Each module is offset from the previous module by 90 degrees making it naturally suited to rectilinear locomotion. The gait is produced by setting the amplitude of the lateral modules to zero and the modules of the vertical axis execute a sine wave [17]. The snake robot is capable of using lateral undulation and to assist the mechanical design, a compliant material made from pads of platinum-doped silicone was added to the outside of the module to provide additional friction and compression. Each module includes a microcontroller to perform internal computation independently of the other modules.

The Unified Snake Robot is the successor to the modular snake robot and was developed by Wright in 2012 at Carnegie Mellon University to be a hyper-redundant, serial-linked snake robot [18]. Each motor has an attached encoder to give feedback on the position of each module. Accelerometers and gyroscopes are used in each module to determine its orientation. The motor current is monitored to allow current limiting to roughly control force output. The Unified Snake robot uses rectilinear locomotion as its primary locomotion gait and can use concertina locomotion for operations such as climbing trees.

Figure 2.7: The Unified Snake Robot climbing a tree (retrieved from [18])

The series elastic actuated snake robot, the SEA robot was developed following the Unified Snake Robot and therefore follows the same design principles like using 1-DOF modules which are offset from the previous module by 90 degrees [19]. An important improvement to this model is the addition of a series elastic actuator in each module to enable compliant motion and fine torque control on each joint. The series elastic actuator allows for more contact with flat ground improving the efficiency of lateral undulation. An IMU containing a 3-axis gyroscope, 3-axis accelerometer and a 3-axis magnetometer was added to each module to allow for more accurate state estimation.

2.3 University of Canterbury Snake Robot

The University of Canterbury is developing a snake robot for use in applications such as industrial inspections or urban search and rescue operations. The snake robot is planned to be a small and low cost designed specifically for operation in unstructured environments. The first and second prototype of the modular snake

Figure 2.8: The SEA snake robot. (retrieved from [19])

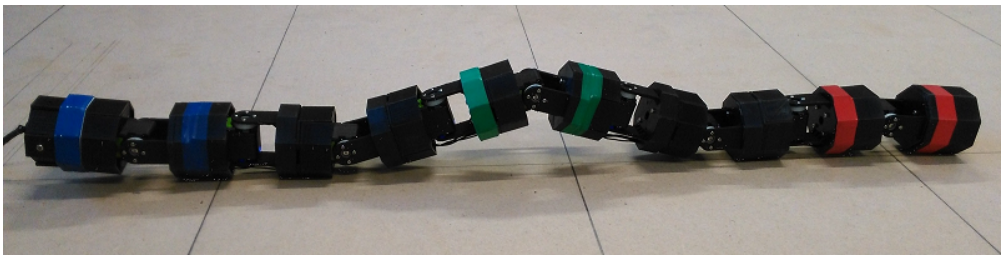
robot was developed by Gilani [1] and was used to test different biologically-inspired gait patterns such as rectilinear locomotion.

2.3.1 Snake Robot Design

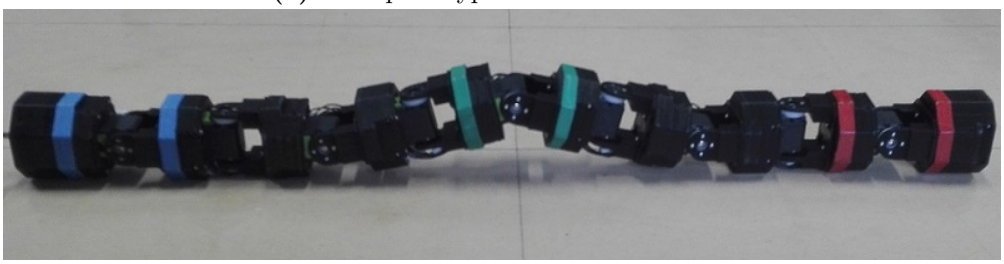
The snake robot consists of a series of ten 3D printed modules, with eight middle modules making up the body between the head and the tail modules. The modules were labelled in order from the head module being module 0 to the tail module being module 9. The robot modules are connected at an offset of 90 degrees from the previous modules such that they are alternatively in the lateral and dorsal plains as inspired by other modular snake robots [16][18]. In the first prototype shown in Figure 2.9a, each module except the head module contained a HerkuleX DRS-0101 smart servo motor, therefore the robot has five yaw joints and four pitch joints.

The head and tail modules are essential for efficient locomotion as they allow the first pitch module to be utilised and makes the length of the last module equal to the rest. The tail module is similar to the body modules but has an extended design to house the slip ring used by the tether. The slip ring is used to prevent the tether from getting twisted while performing the rolling gait. The head module is a passive module which does not contain a servo motor. It was specifically designed to contain a camera to send a video feed to the operator.

The snake robot is powered from a 9 V power supply through a 3m tether running to the tail module. The servo motors use a 5 V TLL serial communication method



(a) First prototype of the Snake Robot



(b) Second prototype of the Snake Robot

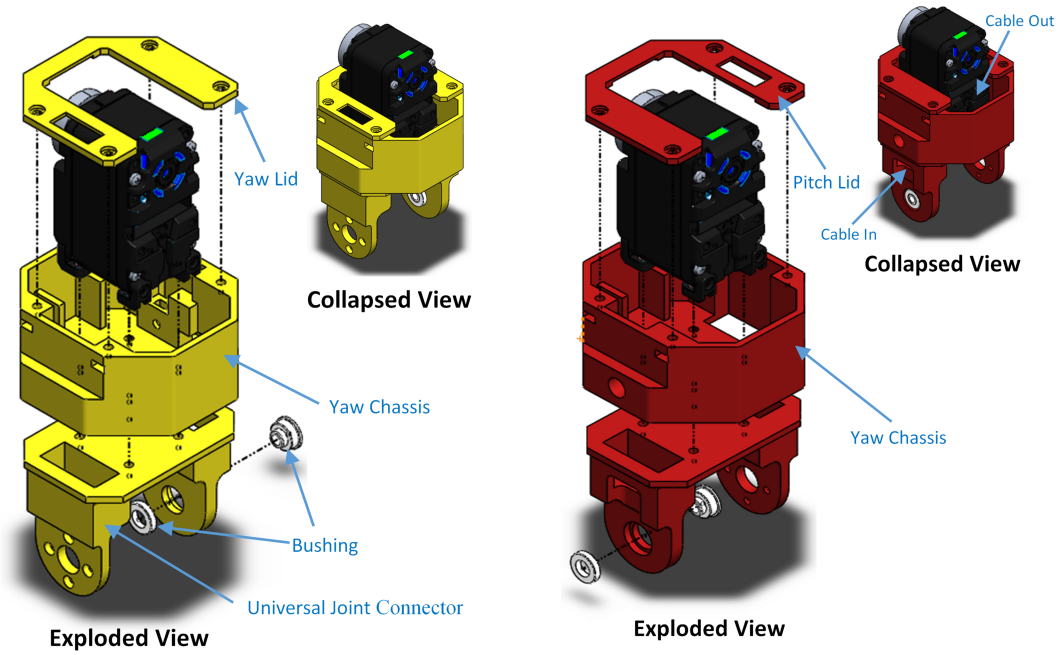
Figure 2.9: The snake robot under development at the University of Canterbury. (retrieved from [1])

from an Arduino Mega 2560 microcontroller connected to the main computer via USB. The servo motors are daisy chained together to allow the control signals to be sent along a single data bus.

The second prototype, which can be seen in Figure 2.9b, was designed to improve some of the issues with the first prototype. The links between the motors were thickened to give them more strength to prevent them from breaking. The modules' length was reduced from 80 mm to 68 mm to improve the manoeuvrability of the snake robot. The modules were split into more parts to enable easier assembly and disassembly. Additional space was included in the design to allow for hardware selected in Chapter 3 to be mounted. The yaw and pitch modules in the second prototype can be seen in Figure 2.10.

2.3.2 Control Design

The modular snake robot is naturally suited to rectilinear locomotion with the modules offset at 90 degrees, however it is also capable of turning, rolling, rotating and sidewinding gaits. A lateral undulation gait is possible but has limited effectiveness with the current robot design. The horizontal friction from the snake robot contacting the ground is too low so there is little forward propulsion generated. The gait patterns are generated from a sinusoid function described in



(a) Yaw module designed by Gilani.

(b) Pitch module designed by Gilani.

Figure 2.10: The yaw and pitch modules of the university of Canterbury Snake Robot. (retrieved from [1])

Equation 2.1.

$$\begin{aligned} \phi(t)_{i,p} &= A_p \sin(\omega_p t + \psi_p(i-1)) \quad i \in \{1...M_p\} \\ \phi(t)_{i,y} &= A_y \sin(\omega_y t + \psi_y(i-1) + \psi_{py}) + O_y \quad i \in \{1...M_y\} \end{aligned} \tag{2.1}$$

Where i is the module number, $\varphi(t)_{i,p}$ and $\varphi(t)_{i,y}$ are the rotation angles of pitch and yaw modules respectively, A_p and A_y control the amplitude of the rotation angle, ω_p and ω_y control the locomotion speed, ψ_p and ψ_p control the shape of the snake robot, ψ_{py} sets the phase difference between the yaw and the pitch function generators, and O_y is the constant angle added to the yaw function generator. The subscripts p and y indicate that the parameters in the pitch and the yaw function generators are independent.

The gait parameters determine the locomotion gait used and the speed of the snake robot. However depending on the gait pattern, the speed may be limited by the torque of the servo motors.

The gait patterns used in this thesis were confined to linear progression, turning, rolling and rotating. The turning gait uses the linear progression gait but adds a yaw offset O_y to turn. The rolling gait is a non-biologically inspired gait which uses two identical sin waves offset by 90 degrees between pitch and yaw joints. The rotating gait is based off the sidewinding gait but only produces rotating motion.

The gait parameters used to produce these gait patterns were developed by Gilani using the V-REP simulator to optimise them and were then verified using the snake robot [1]. The optimal gait parameters found are shown in Table 2.1.

Table 2.1: Optimal gait parameters found.

Gait	A_p	A_y	ω_p	ω_y	ϕ_p	ϕ_y	ϕ_{py}	O_y
Linear Progression	30	0	π	0	144	0	0	0
Turning	30	0	π	0	144	0	0	10
Rolling	30	30	π	π	0	0	90	0
Rotating	10	30	π	π	90	120	0	0

2.3.3 Software Design

The snake robot is controlled using the Robot Operating System (ROS) which was designed as an open-source framework for robotic applications. ROS is a collection of tools, libraries and conventions to simplify complex and robust robot behaviour [20]. It uses multiple processes know as nodes which each perform a specific task. In addition to the user defined nodes, there is a special master node which contains name registration and lookup.

There are two main ways that nodes communicate with each other: topics and services. Topics are an asynchronous system that uses a publisher-subscriber model where a node publishes information with a topic identifying the message content. Any node can then subscribe to that topic to receive the information in the message. Topics are useful for asynchronous events and allow for multiple nodes to access the same information with little overhead. Services are a synchronous system where a node sends a request message and then waits for a reply. The master node tracks publishers and subscribers of topics as well as services.

The snake robot software has three nodes: the gait generation node, the serial communication node and the graphical user interface (GUI) node. The gait generation node running on the main computer generates the motor angles based on

the given gait parameters. The serial communication node is the bridge between the main computer and the Arduino used to control the motors. The GUI node allows the user to control the snake robot. Figure 2.11 shows the nodes and topics used by the three nodes to control the snake robot.

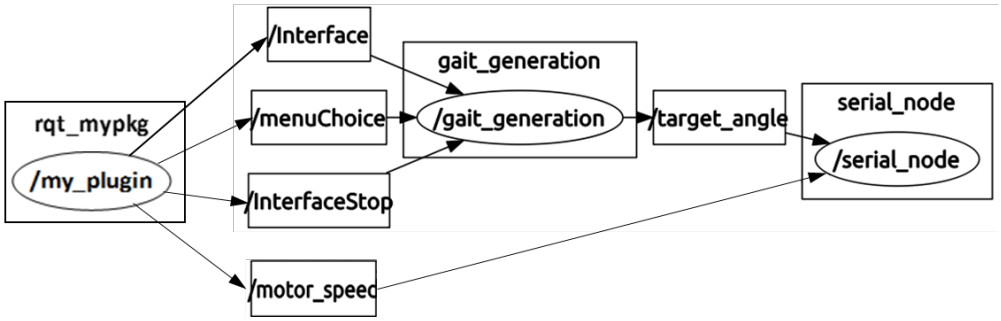


Figure 2.11: ROS communication network used by the snake robot. (retrieved from [1])

The Arduino Mega 2560 microcontroller uses the Arduino platform to program it. The Arduino receives the motor angles through the *"/target_angle"* topic and converts them to TTL serial control signals for the servo motors. An addressing system is utilised where each servo motor has a unique address to control the servo motors on a single data bus. Each servo motor has an inbuilt encoder which sends back data to the Arduino. The control system is using open-loop control such that the signals being sent from the motor encoders are not being utilised.

The graphical user interface (GUI) was created by Gilani to select the gait pattern of the snake robot. The GUI shown in Figure 2.12 has two modes, the first sets the motor angles directly with user inputted angles and the second allows the gait parameters to be selected from a predefined list or directly specified. The encoder angles received from the motors are displayed on the right of the GUI.

2.4 Kalman Filters

The snake robot used open-loop control while the gait patterns were being developed. To autonomously control the locomotion gaits of the snake robot, closed-loop control is needed. In order to create a closed-loop system, a pose estimation algorithm is required, utilising sensor feedback from the robot.

State estimation algorithms have been subjected to a wealth of research. The most common algorithm is the Kalman Filter which is a class of Gaussian Bayes

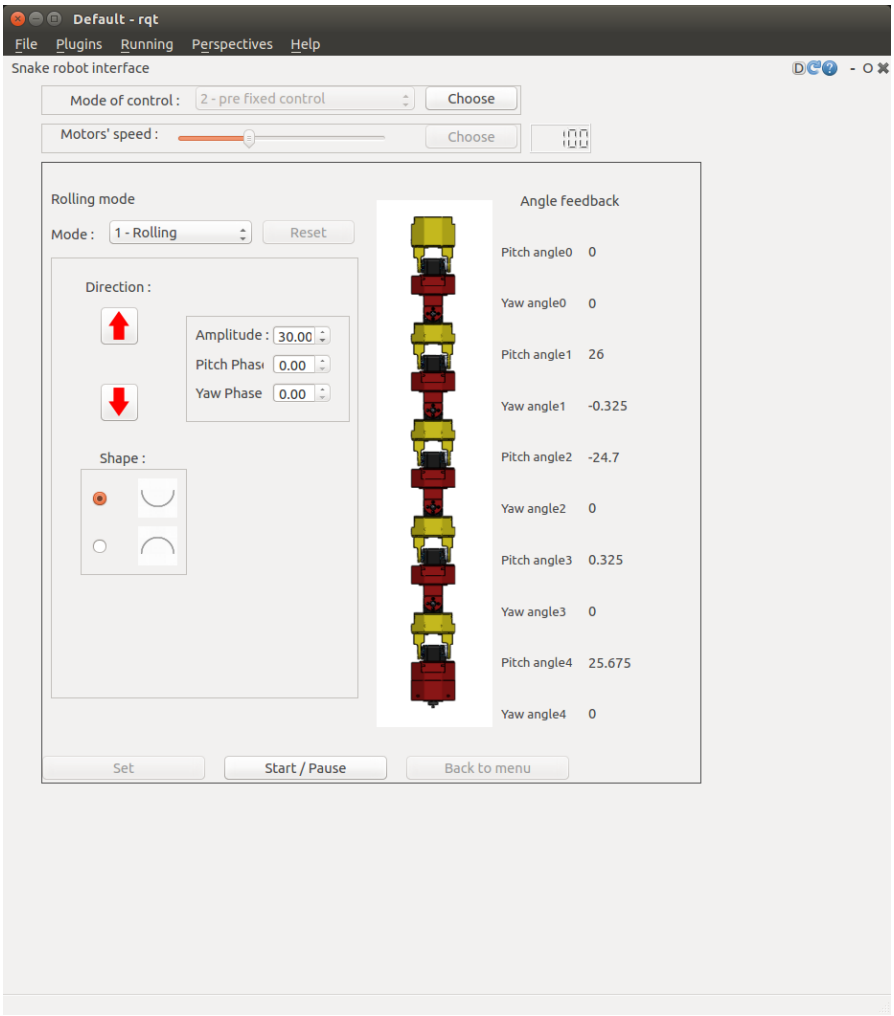


Figure 2.12: The GUI created to control the snake robot.

filter and is a two stage filter in which there is a predict step and an update step [21]. The Kalman filter is recursive and only requires the current state estimate and covariance to be stored between timesteps.

The pose estimation algorithm developed in this thesis is for a non-linear system, therefore a non-linear extension to the Kalman filter is required. The most common non-linear Kalman filter is the extended Kalman filter (EKF) which linearises the system at each timestep by using Jacobians of the process and measurement models in order to calculate the covariance. The EKF is only effective if the system is almost linear on the time scale of the updates [22]. The unscented transform was developed by Julier to overcome this limitation, which is used in the unscented Kalman Filter (UKF) [23].

The UKF uses a deterministic sampling technique using sigma points instead of

Jacobians to calculate the covariance. The $2L + 1$ sigma points, where L is the length of the state vector, are selected based on the square-root deconstruction of the prior covariance [23]. The non-linear function is applied to each of the sigma points to give a cloud of transformed points from which the mean and covariance can be found. Figure 2.13 shows the linearisation of a 1-dimensional system using sigma points to find the UKF Gaussian compared to the actual Gaussian computed from a Monte-Carlo estimate.

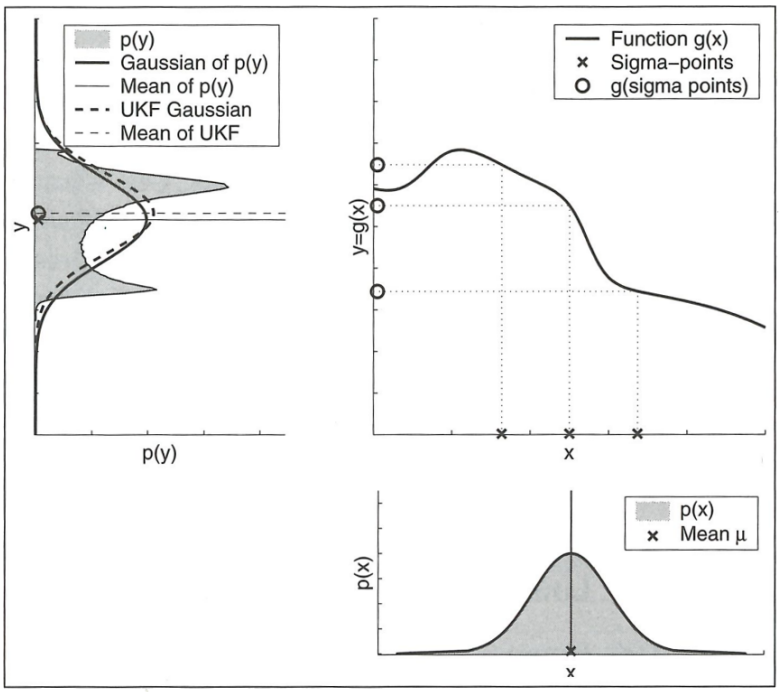


Figure 2.13: Linearisation of a 1-Dimensional system using a UKF with a Gaussian distribution. (retrieved from [24])

Both the EKF and the UKF could be used for pose estimation of the snake robot. Using the UKF eliminates the need to calculate the Jacobian which can be difficult for highly non-linear systems. Studies have shown that the UKF performs comparably to the EKF but has a significantly greater computational cost [25][26]. The UKF has a longer execution time due to using $2L + 1$ sigma points as the UKF therefore has to run the process and measurement model once for each sigma point compared to a single time for the EKF.

Van der Merwe et al. proposed the square-root unscented Kalman filter (SR-UKF) to reduce the computational complexity of the UKF [27]. The SR-UKF has a similar execution time to the EKF with better numerical properties than the UKF and guarantees the positive semi-definiteness of the state covariance.

To reduce the computational expense, the spherically simplex unscented Kalman filter (SSUKF) was developed by Julier [28]. The SSUKF selects the sigma points which lie on the origin or on a hypersphere centered at the origin which reduces the required sigma points to $L + 2$. The SSUKF can be used in conjunction with the SR-UKF for greater efficiency [29]. Lozano et al. compare three variants of the SSUKF and concluded that the SR-SSUKF offers the greatest precision [30]. This thesis uses the SR-SSUKF in the pose estimation algorithm due to the complexity of calculating the Jacobians for the EKF.

2.5 Other Institutions Feedback Systems

Existing snake robots being researched have developed pose estimation algorithms. Using a fusion of accelerometers and gyroscopes is one possibility as described by Rollinson [31]. Equation 2.2 shows the state vector Rollinson proposed using a gait parameter based approach including the error term $\mathbf{e} = [e_1 ... e_n]^T$ for each of the joints in the snake robot, where A is the amplitude of the serpentine curve, ξ is described in Equation 2.3 where v is the temporal component that determines the frequency of the actuator cycles with respect to time, t. $\mathbf{q}$ is the orientation quaternion vector in the world frame and $\boldsymbol{\omega}$ is the snakes robot's angular velocities in the body frame.

$$\mathbf{x} = [A \ \dot{A} \ \xi \ \dot{\xi} \ \mathbf{e}^T \ \mathbf{q}^T \ \boldsymbol{\omega}^T] \tag{2.2}$$

$$\xi = vt \tag{2.3}$$

Rollinson uses an Extended Kalman Filter (EKF) to predict the accelerometer and gyroscope readings allowing the shape of the snake robot to be constructed. It was concluded that they were successfully able to fuse the sensor feedback from the joint angles, accelerometers and gyroscope to estimate the orientation of the robot. The limitation of this model is that it is only designed for lateral undulation gaits. Rollinson compared the use of the EKF to an Unscented Kalman Filter (UKF) and a Spherical Simplex Unscented Kalman Filter (SSUKF) and concluded that the formulation of the estimation problem, filter tuning, and other system-specific considerations were more important factors [32].

The estimated gait parameters are used for compliant control of the robot by commanding gait parameter offsets from the current estimated state [33]. This method worked well for climbing a pipe but is less effective on flat ground due to the limited contact force with the ground. This lead to the development of the SEA Snake module described in Section 2.2.2 with series elasticity to the joints so that the robot is more sensitive to the terrain it contacts.

Later research by Rollinson used a different state vector directly using the joint angles as shown in Equation 2.4, where $\mathbf{a} = [a_x a_y a_z]$ is the robot's world frame acceleration and $\boldsymbol{\theta} = [\theta_1, ..., \theta_n]$ is the robot's joint angles for a robot with m links. Using the joint angles directly removes the requirement of needing knowledge of the gait pattern being performed.

$$\mathbf{x_k} = \left\lfloor \mathbf{a}_k \ \mathbf{q}_k \ \boldsymbol{\omega}_k \ \dot{\boldsymbol{\omega}}_k \ \boldsymbol{\theta}_k \ \dot{\boldsymbol{\theta}}_k \ \ddot{\boldsymbol{\theta}}_k \right\rfloor \tag{2.4}$$

Zhang proposes a similar method using gravitational and gyroscopic sensors in a snake-like robot used for surgery [34]. The gyroscope is used to predict the rotation difference by orientation integration. A complementary filter is used to update the prediction from the orientation difference derived from the consecutive acceleration vectors. It was concluded that using this method, accurate joint angle estimation could be achieved.

Most snakes main propulsion gait is lateral undulation in which they use horizontal friction to propel themselves forward. The irregularities in the terrain around them are used as push-points to increase their efficiency [35]. For a biologically inspired snake robot, the horizontal friction from the underside of the snake robot touching the ground must be greater than the friction in the links such that the robot is propelled forward. Mimicking the same kind of obstacle-aided propulsion is an important current research into lateral undulation control. In an unstructured environment, there are a lot of obstacles that would need to be either avoided or used to aid the robot. By using obstacles for propulsion, efficiency of the snake robot can be improved.

Liljebäck proposed a hybrid model for the dynamics of a snake robot interacting with obstacles [36]. A linear complementarity problem is used to solve for the contact forces on an obstacle. A hybrid controller was proposed to enable the snake robot to propel itself forward using obstacles in the environment. Experimental

results using the proposed hybrid controller was conducted using the Kulko snake robot in [37]. The results from the experiment showed that the proposed controller successfully maintained the forward propulsion of the snake robot in the considered obstacle courses.

2.6 Summary

Biological snakes have inspired the development of snake robots due to their efficiency of locomotion through complex environments. Snakes gait patterns can be categorised into four types: lateral undulation, sidewinding, concertina and linear progression. These gaits have been replicated by early snake robots starting with Hirose's ACM series in 1979 which used passive wheels [38].

More recent snake robots include the work conducted at the Carnegie Melon University on hyper-redundant, serial-linked snake robots. The modules of these snake robots are offset from each other by 90 degrees making them naturally suited to linear progression, though they are capable of other gaits. The later robots like the SEA snake contained accelerometers and gyroscopes as well as the motor encoders to further allow for pose estimation to be performed.

The University of Canterbury snake robot is being developed as small, low-cost robot for operation in urban search and rescue and industrial inspections. The mechanical design has ten modules and was developed by Gilani [1]. The modules are connected at an offset of 90 degrees creating five yaw modules and four pitch modules. Each module apart from the head module contains a HerkuleX DRS-0101 smart servo with room for additional electronics. The head module is passive and was designed to carry a camera to give visual feedback to the operator.

The robot is powered through a 3 m tether from a power supply to the tail modules which has a slip ring to prevent the tether from getting twisted. The motors are controlled through a TTL serial communication bus from a Ardunino Mega 2560 microcontroller connect to the computer via USB.

The gait patterns used by the snake robot are generated from a sinusoid function shown in Equation 2.1. The gait patterns used in this thesis are linear progression, turning, rolling and rotating. The gait parameters to produce these gaits were optimised by Gilani and can be seen in Table 2.1 [1].

The snake robot is controlled using ROS which is an open-source framework for robotic operations. Three nodes are used to control the robot, the gait generation node, the GUI and the serial communication node. Topics are used between each of the nodes to allow the robot to be controlled from the GUI.

Kalman filters are commonly used to fuse sensor data for pose estimation. The snake robot pose is non-linear so a non-linear extension needs to be used. The EKF is the most common non-linear filter, though it requires the Jacobian to be calculated. The UKF can instead be used which uses a deterministic sampling approach called sigma points. The UKF is gives similar results to the EKF but is much more computationally expensive.

The SR-UKF and the SSUKF were both developed to reduce the computational complexity of the UKF. The SSUKF reduces the number of sigma points from $2n + 1$ to $n + 2$, where n is the number of states, by choosing points on the the origin or hypersphere centered at the origin. Both of these approaches can be used in conjunction to make the computational efficiency greater than the EKF.

Rollison uses both a gait based and joint angle approach to pose estimation of the snake robots developed at Carnegie Melon University. Rollison used the EKF, UKF and SSUKF and concluded that they all perform comparably and the choice of the estimation problem and filter tuning were more important.

Chapter 3

Hardware Design for Sensor Data Collection

The modular snake robot designed by Gilani contained just a Herkulex DRS-0101 Smart Servo on board each module used for locomotion [1]. The smart servo contains a motor encoder to give feedback on the motor angle of each module. For pose estimation of a snake robot such as proposed by Rollinson, a three axis accelerometer and a two axis gyroscope are required as well as the motor encoder data [31].

The sensor data is processed remotely on a desktop computer so therefore the sensor data must be communicated to the computer. This is achieved via a tether, which also provides power to the robot meaning the robot is not required to carry batteries. The motors used TTL serial communication through the tether from an Arduino Mega 2560 microcontroller connected to the main computer.

This chapter covers the selection of the hardware required to develop a pose estimation algorithm. A low-cost inertial measurement unit (IMU) was selected to provide the orientation of each individual module. A data bus was selected to send sensor data from each of the robot modules to the computer through the tether. A microcontroller is used to interface with the IMU and connect to the data bus.

3.1 IMU Selection

The pose estimation algorithm developed is based on the method by Rollison [31] which requires an accelerometer and gyroscope on each module. Recent advancements in IMU technology means there are a number of nine degrees of freedom (DoF) IMUs available, each containing a three axis accelerometer, a three axis gyroscope and a three axis Magnetometer.

A common problem with IMUs is sensor drift where small errors in the position calculation accumulate over time from integration. In other IMU applications such as on UAVs, an additional sensor like a GPS unit is used to correct for drift by giving the IMU an additional reference to correct against. The snake robot is designed to operate in complex environments where there is no guarantee that GPS could operate and as such is not a viable option. Madgwick proposed that a magnetometer could be used to provide the additional reference required to help counter the sensor drift [39]. As the servo motors generate a magnetic field, the magnetometer readings are unreliable and therefore cannot be used in those modules. However the head module is passive and does not contain a servo motor, therefore a head-mounted magnetometer could be used in the Madgwick filter.

A low cost IMU is desirable for the snake robot as one is required by each module. The low cost IMUs available shown in Table 3.1 are predominantly manufactured by two companies, STMicroelectronics and Invensense. The 9 DoF IMUs generally contain multiple sensors from the same manufacturer on a specifically designed carrier board. The individual sensors used on the various carrier boards from each company were compared to determine which sensors were best for the snake robot which can be seen in Appendix A.

Table 3.1: IMUs available using the researched components.

Model	**Accelerometer**	**Gyroscope**	**Magnetometer**	**Cost (NZD)**
MinIMU-9 v5	LSM6DS33	LSM6DS33	LIS3MDL	18.70
LSM9DS1 9DoF	LSM9DS1	LSM9DS1	LSM9DS1	32.71
IMU 10 DoF	MPU9255	MPU9255	MPU9255	19.72
Freetronics 9 DoF	MPU9150	MPU9150	MPU9150	52.33

The IMU 10 DoF and the Freetronics 9 DoF both contain MPU IMUs which have good accelerometers and gyroscopes but contain considerably worse magnetometers than the other IMUs considered. The magnetometer is being used to correct

sensor drift in the head module so a good magnetometer is preferred. The LMS9S1 9DoF and the MinIMU-9 v5 has a similar quality of magnetometer and gyroscope but the MinIMU-9 v5 has a better accelerometer due to being a newer sensor. The MinIMU-9 v5 is also considerably cheaper than the LSM9DS1 and was therefore selected for developing the pose estimation algorithm on the snake robot.

The IMUs were calibrated after being mounted on the snake robot to account for any misalignment errors. The accelerometers and magnetometers were calibrated using a least squares method proposed by Ammann et al. [40].

3.2 Data Bus Selection for Robot Tether

The sensor data from each IMU needs to be sent to the main computer to be processed by the pose estimation algorithm. The data could be returned to the computer using either wireless communication or through a data bus. The snake robot was designed to be powered through a tethered power supply so adding additional wires for data communication would not affect the functionality of the snake robot. Using wireless communication adds complexity to the system where it has to reliably maintain connection to the main computer regardless of the environment. As a data bus through a tether is easier to implement and more reliable for accurate communication while designing the pose estimation algorithm this is what was chosen for data communication. For practical use, the tether would have to be of sufficient length to operate in industrial environments, and so the tether is expected to be able to operate at up to 100 m. The data bus protocol must therefore be able to support this length.

The MinIMU9-v5 can output data using either serial peripheral interface (SPI) or Inter-Integrated Circuit (I^2C) protocols to output data. SPI is a single-master, multiple-slave unbalanced serial protocol designed for short range communication between integrated circuits. I^2C is a multiple-master, multiple-slave serial communication bus designed for short distance or intra-board communication typically between a processor and peripherals. Though both SPI and I^2C are able to reach lengths of 100 m under the right conditions [41][42], there are protocols designed for the purpose of long range communication that are more suitable for use with the tether. A microcontroller is therefore required to receive the I^2C from the IMU and send the data through the selected data bus.

Two suitable, commonly used options are RS485 and the Controller Area Network (CAN) bus. RS485 is a hardware standard designed for high speed, long distance data buses of up to 1200 m. RS485 is a two wire differential signal serial bus that can be connected to a microcontroller on each module allowing for half-duplex operation. The RS485 also supports a four wire connection to allow for full-duplex operation if desired. Most microcontrollers do not support RS485 so an additional integrated circuit to translate from the microcontroller's UART (selected in Section 3.3) to RS485 is needed. The RS485 standard can operate at up to 50 Mbps at 1m and 100 kbps at 1200m [43], though the speed can be increased using techniques such as preemphasis [44]. At 100 m, the RS485 standard can operate at up to 7 Mbps depending on the conditions as shown in Figure 3.1. Being a hardware standard, messaging protocol is controlled at a software level. RS485 is therefore reliant on the software for collision detection and error checking.

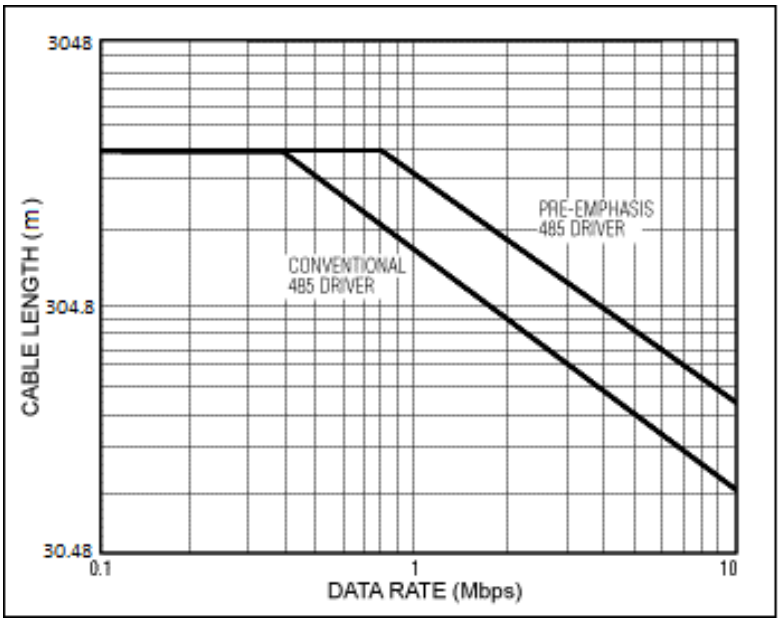


Figure 3.1: Data rate versus the cable length of the RS485 Standard. (adapted from [45])

CAN bus is a full protocol including both the hardware layer and the data link layer. It uses a two wire differential signal like RS485 for the hardware layer supporting half-duplex operation. The data link layer is a multi-master protocol that uses a prescribed data frame including the message identifier, data and control bytes. The protocol can operate at 1 Mbps up to 40 m and can operate at 125 kbps at 1600 m [46]. At 100 m, the CAN Bus can operate at approximately 600 kbps as shown in Figure 3.2. Being a full protocol, CAN bus supports collision detection, failure detection and automatic resending of disrupted packages. Most

microcontrollers do not support the CAN bus so a CAN controller and transceiver is required to interface with them.

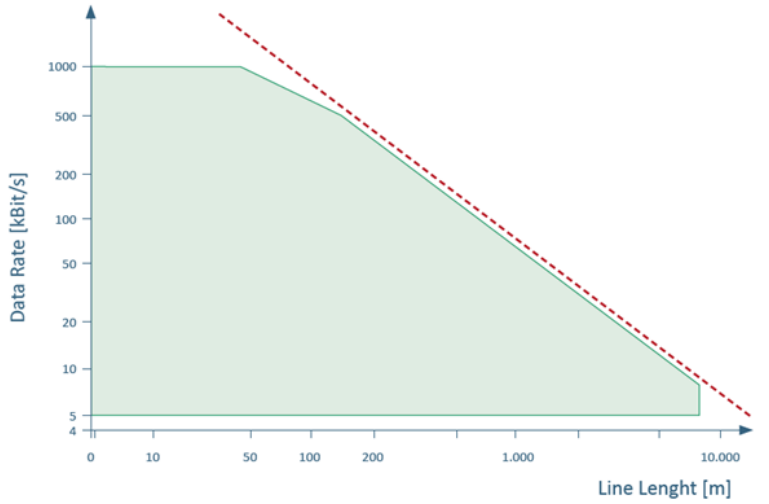


Figure 3.2: Data rate compared to the cable length of the CAN Bus. (retrieved from [47])

The RS485 hardware standard was selected as the data bus for the snake robot from the microcontrollers inside each module through the tether to the base station due to being easier to implement and offering a faster bit rate. The RS485 standard uses twisted pair wires with a characteristic impedance of 100-120 ohms and can be optionally shielded. Shielding is recommended for longer cable lengths, however this was decided against as it caused the wire to be too rigid affecting the operation of the snake robot during testing. The twisted pair is terminated at each end of the wire using resistors equivalent to the characteristic impedance of the wire. Additional pull up and pull down resistors were connected at the start of the wire. A third wire carrying a common ground is required for the operation of the RS485 standard. The stub lengths inside each module are kept short to prevent the stub acting as a separate transmission line. Figure 3.3 shows the data bus connections between the Master and slave devices and the required resistors.

An unshielded cat-5e cable with a characteristic impedance of 100 ohms was selected for the RS485 data line, which follows the RS485 standard. RS485 only requires two of the eight wires in a cat-5e cable so two of the other cables were used for the servo motor signal lines to reduce the number of cables in the tether. Separate power and ground lines were used to protect against inductance affecting the data lines.

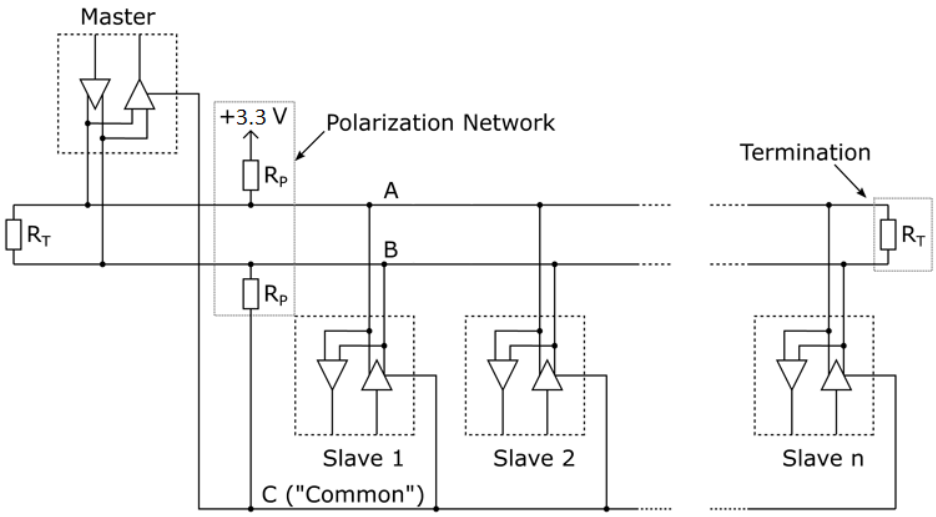


Figure 3.3: Schematic of the RS485 data bus connections between the Master and Slave devices and the required termination resistors. (adapted from [48])

3.3 Microcontroller Selection

Each module requires a microcontroller in order to gather the sensor data from the IMU to send it via the RS485 communication to the main computer. The snake robot as designed by Gilani had no microcontrollers inside each of the snake robot modules and the servo motors were controlled by an Arduino Mega connected to the main computer [1]. An Arduino Mini Pro had been selected as a microcontroller to go into each module due to fitting in the small space allocated for sensors. The Arduino Mini Pro was used in early testing of the IMUs and the RS485 data bus.

The Arduino Mini Pro connects to the computer through an FTDI connector which shares the same serial port as serial 1. Sharing a connection means that data cannot be outputted to the computer for testing while serial 1 is being used. However the mini pro only has a single serial port so the RS485 data bus had to use serial 1. The shared serial port made developing software for the microcontroller using RS485 difficult so a more suitable microcontroller that could be used on each of the snake robot modules was researched.

A suitable microcontroller had to be small enough to fit inside the snake robot module, have enough serial ports for simultaneous RS485 communication, computer connection and servo connections, and have I^2C or SPI ports for the IMU. A

faster microcontroller using the Arduino platform was desirable to make porting existing software trivial and allow for fast processing of sensor data and algorithms.

The alternative selected for the microcontroller in each module was the Teensy LC. The Teensy LC has a 48 MHz Cortex-M0+, 3x faster than the Arduino Mini Pro and supports the Arduino platform. It has three serial ports with an additional port dedicated to a micro USB connection to a computer. It meets the requirements of being small enough to fit inside the Snake Robot module and can interface with the servo motor, RS485 communication and sensors. The main drawback to the Teensy LC is that it is a 3.3 V device compared to the 5 V device of the Arduino Mega.

Two possible solutions are to use a logical level converter for the RS485 data bus or to select a different microcontroller for the base station device. The Arduino Mega 2560 was replaced with a Teensy 3.2 to keep consistency between all the microcontrollers used for the snake robot. The Teensy 3.2 full version of the Teensy LC with a 72 MHz Cortex-M4 and is also a 3.3 V board so can easily interface with the other Teensy LCs. The Teensy 3.2 does not have a 5 V output so a logic level converter (LLC) is required to communicate with the servo motors which use 5V signals. The Teensy microcontrollers do not support the RS485 standard so a MAX3485 integrated circuit was used to connect the UART to the RS485 data bus. The overall layout of the hardware selected can be seen in Figure 3.4.

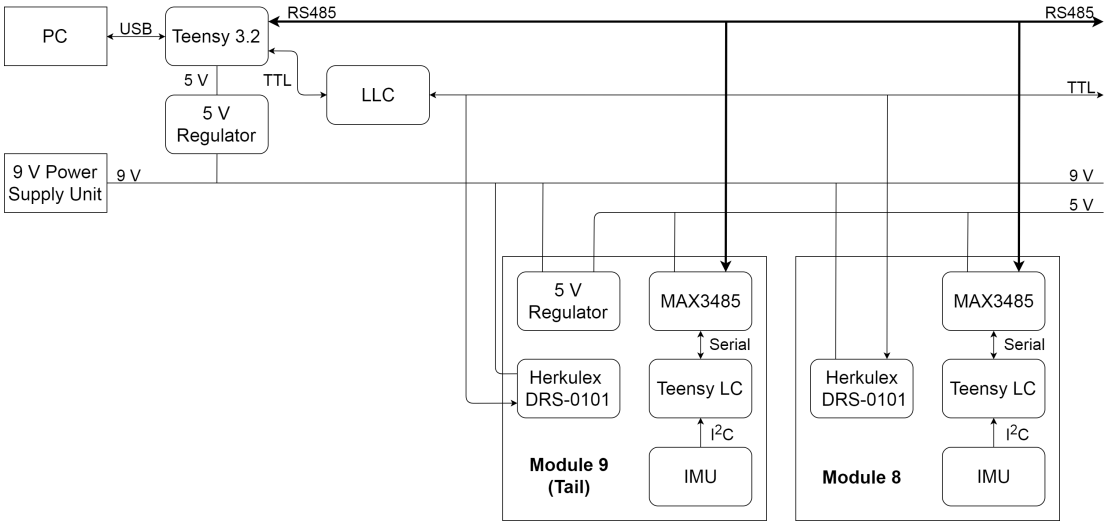


Figure 3.4: Layout of the hardware to collect sensor data from the snake robot.

3.4 Module Redesign

To accommodate for the change of microcontroller and the electronics required, the snake robot modules had to be redesigned. Gilani took the electronics into consideration when designing the second prototype of the modular snake robot [1]. Each module of the new design contains a Teensy LC, RS485 board, MinIMU-9 v5 and the Herkulex DRS-0101 Smart Servo as well as the wires to connect them. A path for the wires between the modules was built in to prevent connections from interfering with the motor actuation. A four wire connector was used for the RS485 data bus between each module to allow them to be individually disconnected. Access to each of the Teensy LCs micro-USB port was included in the design for testing proposes. The tail module contains the 5 V voltage regulator to supply power to the Teensy LCs from the main power line. Figure 3.5 shows the redesigned module wired up with the selected hardware.

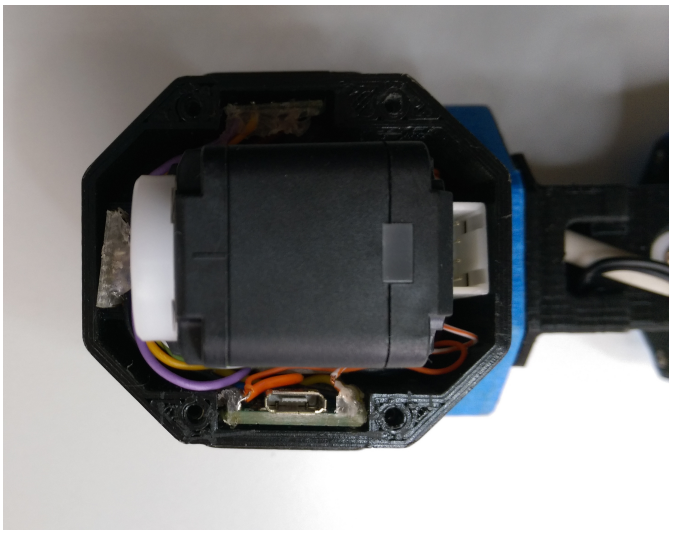


Figure 3.5: Layout of the hardware in a snake module.

3.5 Base Station Design

The base station was designed to be the intermediate point between the computer and the components on the snake robot. It contains the Teensy 3.2 and the additional components it requires to communicate with the snake robot. The Teensy 3.2 is a 3.3 V microcontroller while the Herkulex servo motors use a 5 V signal. A LLC was therefore included to communicate with the servo motors. A MAX3485CSA integrated circuit was included for the RS485 data bus to the

Teensy LCs which is connected to the PCB through an Ethernet port. To allow the entire snake robot to be powered by a single power source, a 5 V regulator was included that is connected to the 9 V power supply. A PCB was designed and fabricated with each of these components to create a portable base station which can be seen in Appendix B.

3.6 Summary

The first modular snake robot design by Gilani contained just a Herkulex DRS-0101 Smart Servo in each module [1]. Each servo motor contains a encoder which is used for pose estimation. Additional hardware was selected to develop the pose estimation algorithm.

The MinIMU9-v5 was selected as the IMU as it has newer sensors and is low cost, and each was calibrated using a least squares method after being mounted on each module [40]. As a data bus was required to return the sensor data from the robot modules to the main computer to perform pose estimation, the RS485 hardware standard was selected due to being a high speed and long range data bus which is needed for the expected tether length of 100 m. With a tether of this length, RS485 has a speed of up to 7 Mbps. As the MinIMU9-v5 does not support RS485, the Teensy LC was chosen to be incorporated into each module. The Teensy LC retrieves the IMU data and sends it through the RS485 data bus via a MAX3485CSA integrated circuit to a Teensy 3.2 connected to the main computer. To accommodate the additional electronics the robot modules were redesigned to house the components and to give easy access to the Teensy LC micro-USB port.

Chapter 4

Synchronising Sensor Data

There was a problem with the sensor collection method of the first hardware design described in Chapter 3 where the motor encoder data did not match the commanded position. Table 4.1 shows an example of the motor encoder data received compared to the commanded position the motors were expected to reach. It can be seen that the motor encoders data is not in sync with each other.

Table 4.1: The commanded angle of all the motor angles compared to the Encoder angle.

Motor	**Commanded Angle (Degrees)**	**Encoder Angle (Degrees)**
1	17.65	12.35
2	-24.30	-27.95
3	17.65	13
4	-24.30	-26
5	17.65	14.3
6	-24.30	-25.35
7	17.65	14.625
8	-24.30	-24.375
9	17.65	17.225

The sensor data is collected from two different sources: the IMUs, and the motor encoders. The IMU data is collected by a localised Teensy LC on each module at a much faster rate than the main computer requests it. When data is requested, a RS485 message is sent to all Teensy LCs to save the currently recorded IMU values; therefore they are in sync with each other. The motor encoder data is collected by the Teensy 3.2 which is connected directly to motors which are daisy chained together through a TTL connection along with power from a power supply

unit. Each motor contains an internal encoder and motor driver which can be accessed through Herkulex commands such that each motor is effectively a black box. Encoder data has to be collected individually from each motor due to the limitations of the Herkulex Smart Servo.

The sensor collection system and other pose estimation software was analysed to determine each tasks execution time. The order that the snake robot software operates in, which can be seen in Figure 4.1, was designed to give the pose estimation algorithm the sensor data at the end of each motor actuation cycle. The sensor data is collected at a rate of 10 Hz which triggers each subsequent task. The software design executes each task as soon as the previous task has been completed regardless of the time taken, so if one task has a long execution time, all subsequent tasks are delayed until it finishes. The execution time of each task was determined by outputting the system time at the end of task. Figure 4.2 was created using the execution times with the assumption that the Teensy 3.2 always succeeds in retrieving the motor angles where time 0 is the start of each execution cycle. It can be seen that the collection of the sensor data has the longest execution times due to communicating with the motors and the Teensy LCs. The motors are assumed to have a 100% actuation time with the commanded angle being reached at the end of the cycle which can be seen in Figure 4.2 next to the execution time.

From Figure 4.2 it can be seen that the sensor collection takes a significant portion of the actuation cycle. The encoder angle retrieval in particular takes a total of 18 ms to execute. As each encoder angle has to be collected individually, each is collected 2 ms after the previous encoder. The long execution time of the sensor collect means that the motors are still moving while the sensor data is collected, which explains why the encoder data are not in sync with each other. The motor data recorded in Table 4.1 had a 70% actuation time which accounts for why the motors 8 and 9 reach the commanded angle contrary to expected from Figure 4.2. Sending the motor angles to the motors experiences the same problem as the data retrieval with each angle taking 1 ms to be sent. The motor angles are therefore sent across a 10 ms time period such that the motors are not operating in sync with each other.

The Teensy 3.2 does not support multitasking, therefore the IMU values are being recorded directly after the encoder values are retrieved, with each Teensy LC recording the data at the same time when the initial save command is received.

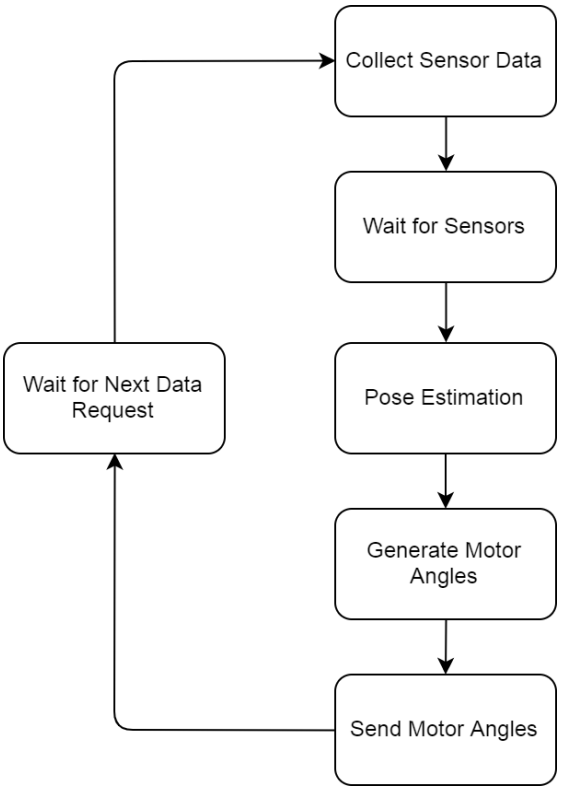


Figure 4.1: The operation cycle of the snake robot software.

The IMU data are therefore all in sync with each other with approximately the same time stamp as motor 9.

4.1 Solutions to Synchronise the Sensors

The sensor data coming from the two different sources which are not in sync prevents the model being used by the pose estimation algorithm from working correctly. Two possible solutions to solve the issue are to integrate control of the motors into the Teensy LCs in each module or use software synchronisation to correct the encoder data.

Integrating control of the motors would allow the encoder data to be collected simultaneously across all the motors and at the same time as the IMU data which would reduce the data collection time. For 100% actuation time of the motors, the sensor data would be recorded before the motor reaches its commanded angle as shown in Figure 4.3. As the data would be in sync, the pose estimation algorithm model would function correctly giving the pose at the time the sensor data was

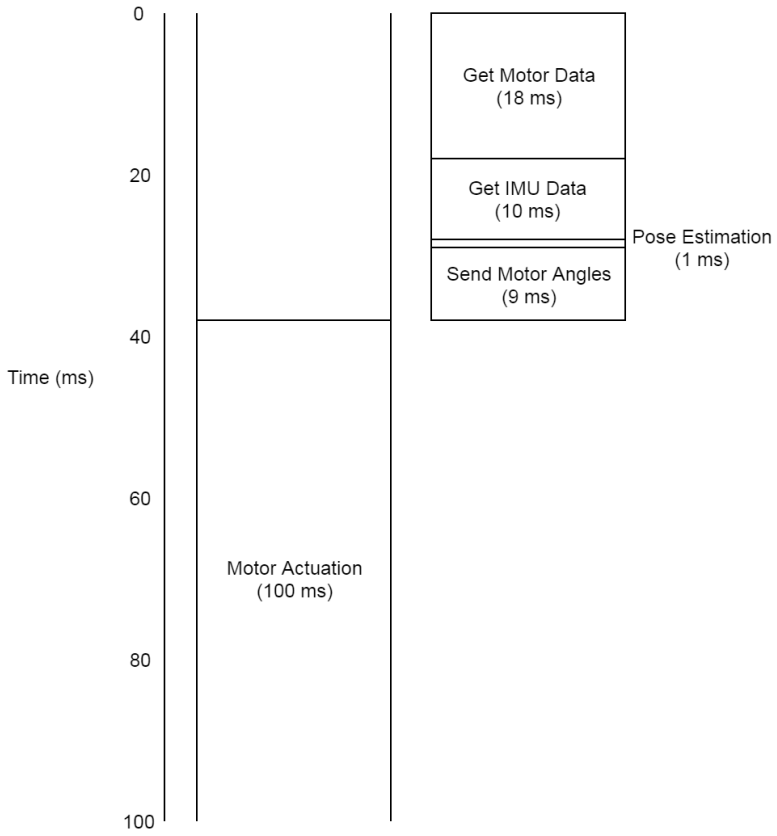


Figure 4.2: Motor actuation time compared to the sensor reading time.

recorded. A LLC would be installed into each module to allow the Teensy LCs to communicate with the motors which require a 5V signal. Motor commands would be sent through the RS485 data bus to the Teensy LCs. The encoder data would be included with the IMU data and sent back via the RS485 data bus. Using the hardware solution, additional sensors can be easily added to each module using existing software and communication system.

The second proposed solution is to use the current hardware setup and use software synchronisation to sync the encoder data with the IMU data. The discrepancy in the commanded position and the encoder data as shown in Table 4.1 can be modelled using the known motor profile and the time stamp at which each reading is taken. The motors use a trapezium profile as shown in Figure 4.4 where the motor accelerates and decelerates 20% of the movement time. The maximum speed is determined by the distance the motor is required to travel in the allocated time. To accurately use the motor profile described, the timestamps of when the motor

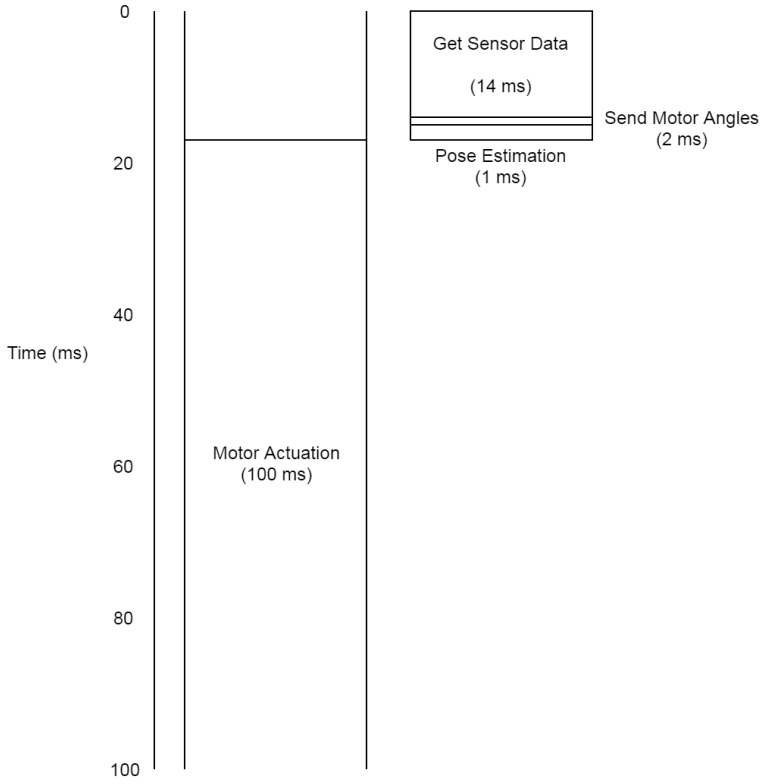


Figure 4.3: Expected execution time of the sensor collection using the hardware solution.

command was sent and when the sensor data was received would be required.

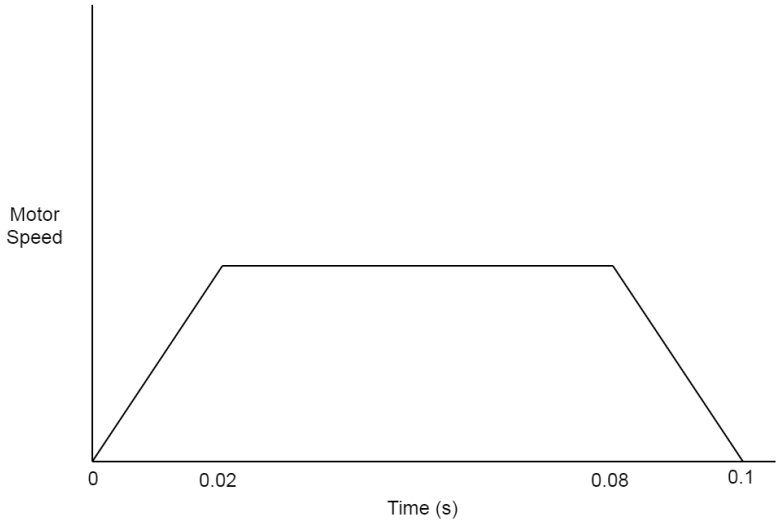


Figure 4.4: Motor profile used by the Herkulex DRS-0101 Smart Servos.

Sensor synchronisation is a topic that has been extensively researched due to the prevalence of large sensor networks that require precise tracking of events such

as sensor data. There are many proposed algorithms for synchronising data from different sensor nodes which use different approaches to maintain a global clock. A master-slave approach is the most common approach where one node is the master and all other nodes are synchronised to it, such as the continuous clock synchronization in wireless protocol proposed by Mock et al [49]. An external clock source that is available to all the nodes such as GPS can also be used to achieve clock synchronisation [50]. The clocks on each node can be corrected continually to keep the network in sync as used in the Timing-sync Protocol for Sensor Networks proposed by Ganeriwal et al. [51]. Alternatively the sensor nodes local clocks can be untethered such that each node maintains its own clock and local timestamps are compared to a lookup table to maintain a global time, which is the system used in the Reference-Broadcast Synchronisation algorithm proposed by Elson et al. [52].

ROS uses the computer system clock as the global clock for the nodes running on the computer. The node running on the Teensy 3.2 is updated using the system clock when messages are received so that encoder data can be timestamped. The ROS serial package has clock synchronisation using a service to update the time when a message is sent in each direction [53]. As the Teensy LCs on the snake robot are not connected to the ROS system, they do not have access to the system clock and therefore cannot time stamp without synchronising the local clocks. The time at which the sensor request is sent to the Teensy LCs could be recorded and stored in the header of the ROS message containing the data from the Snake Robot.

4.2 Sensor Synchronisation Implementation

The improved hardware system with the motors using a 100% actuation time was chosen to solve the sensor synchronisation problem. Improving the hardware system was easier to implement from a software standpoint and does not rely on all the motors functioning correctly. If additional sensors are added to the modules in future research, having all the hardware controlled from the same microcontroller is desirable to make the snake robot more modular. The new hardware design implemented can be seen in Figure 4.5.

A new test was conducted to compare the motor encoders data to the commanded

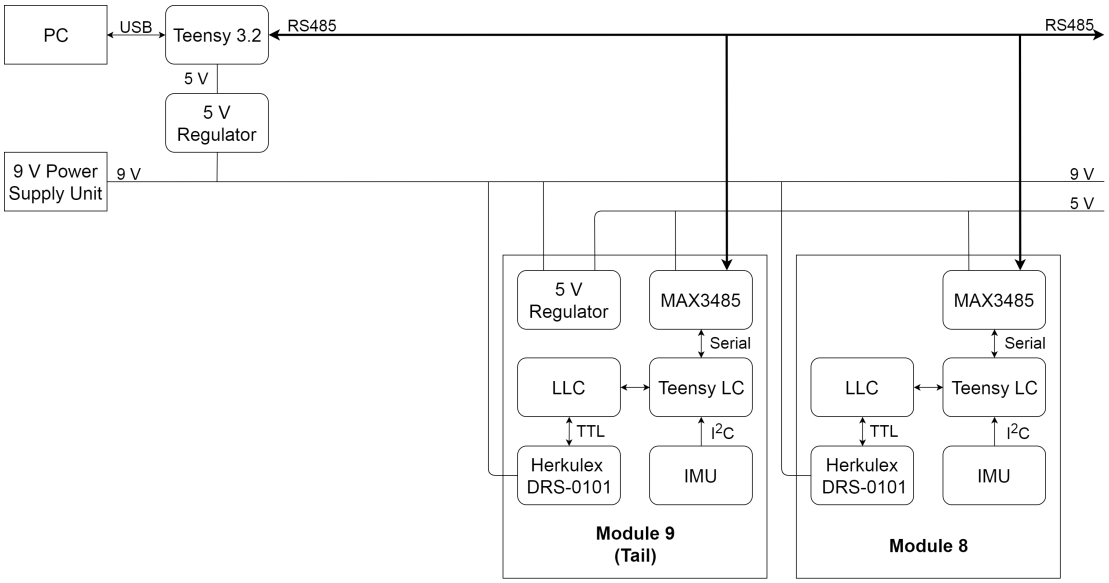


Figure 4.5: New hardware design to collect sensor data from the snake robot.

position to determine the effectiveness of the new hardware design. Table 4.2 shows that with the new hardware, the motor encoders are in sync with each other such that all of the yaw modules and pitch modules have similar joint angles. However, all of the motors are lagging substantially behind the commanded position. The sensor data is collected approximately 20 ms before the end of the actuation cycle, as shown in Figure 4.3, but this does not fully account for the error in the motor angle. The motors were profiled using the Teensy LCs to further investigate the motors lagging behind the commanded position.

Table 4.2: The commanded angle of all the motor angles compared to the Encoder angle with the new hardware design.

Motor	**Commanded Angle (Degrees)**	**Encoder Angle (Degrees)**
1	17.65	11.7
2	-24.30	-28.6
3	17.65	11.7
4	-24.30	-28.275
5	17.65	11.05
6	-24.30	-27.95
7	17.65	10.725
8	-24.30	-27.3
9	17.65	10.725

4.2.1 Motor Profile

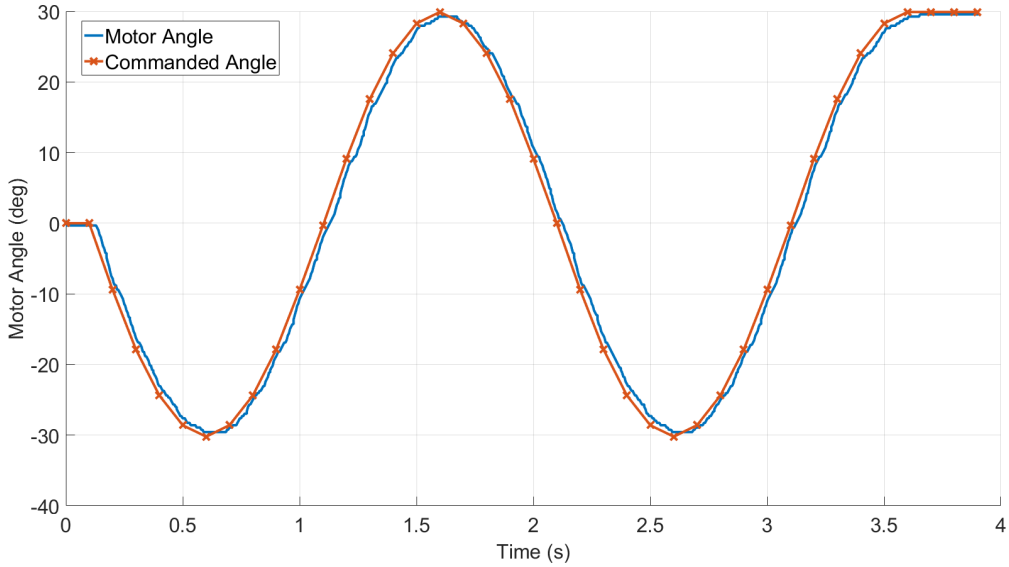
The Herkulex DRS-0101 used in each of the modules were profiled by taking data readings at rate of 200 Hz using the Teensy LC on each module. The motor angle data from the internal encoder were recorded while performing a rolling gait over one complete cycle. The internal time that the motor command and data request were received was recorded to compare the target angle with the encoder angle. The motor angles recorded by the encoders were plotted against the command angle shown in Figure 4.6.

Figure 4.6 compares the motor profile of the motor in the tail module (module 9) to the motor in module 1, which is the first motor at the head of the snake robot. It can be seen that motor 9 is closer to the commanded angle than motor 1. From test data of each of the other motors, the difference between the commanded angle and the motor angle gets larger the further down the daisy chain the motor is.

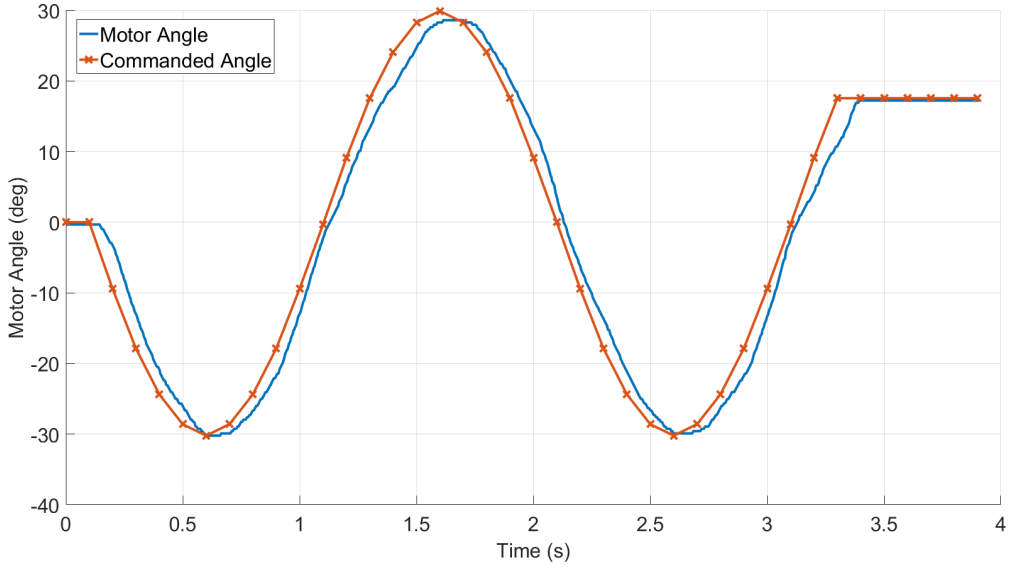
It is suspected that the cause of this is the servo wires connecting the motors together restricting the power drawn by the motors which would explain noticeable delay between the tail and head moving during the initialisation of the snake robot. To improve the motor response, hardware changes to the power distribution are required.

From the test data, the request for sensor data occurs between 0.075 and 0.080 seconds after each motor command as expected. This delay and the delay caused by the power distribution system explains the difference between the sensor data and commanded angle. To decrease the impact of the delay between the motor angle and the commanded angle, the speed of the snake robot was halved such that a complete cycle of the gait pattern takes four seconds instead of two seconds.

The pose estimation algorithm developed in this thesis predicts the joint angle based on the sensor data not the commanded position such that the pose estimated will be that of the sensor data. For the sake of pose estimation, this is the expected behaviour, however if pose estimation is used for control purposes the delay between the commanded position and the pose estimation needs to be taken into account.



(a) Motor profile of motor 9 which is closest to the tail of the robot where the tether is located.



(b) Motor profile of motor 1 which is furthest from the tail of the robot.

Figure 4.6: Comparison of yaw motor profiles at the head and tail of the snake robot with motor commands being sent every 100ms.

4.3 Summary

The sensor data retrieved from the snake robot did not match the commanded position due to the daisy chained Herkulex Smart Servo motors. LLCs were added to each module so that the Teensy LCs could send commands and retrieve encoder data from the Motors. The new system solved the issue of the encoder data being out of sync with each other and the IMU data. Testing the new system revealed

that there is a lag between the commanded position and the motors reaching that position. It is thought that this is due to the power distribution system being unable to provide the required power. The pose estimation algorithm is not affected by this as the sensor data is in sync, however for future control applications it should be taken into account.

Chapter 5

Pose Estimation Algorithm Design

To make intelligent control decisions for the snake robot, a feedback system is required for closed-loop control. As discussed in Chapter 3, the motors have encoders to give feedback on the joint angles between modules, and IMUs in each module to give their orientation. The sensor data was used to develop a pose estimation algorithm that can track the orientation of each module and of the snake robot as a whole. The algorithm is designed to be robust so that modules failing or sensor data errors do not adversely affect the accuracy of pose estimation.

This chapter presents the pose estimation algorithm developed based on the gait patterns developed by Gilani [1]. Each gait pattern uses a different set of parameters, so a model for each is outlined. A model using joint angles instead of gait patterns was created to compare the two different approaches. A virtual chassis is used to abstract away the internal shape of the robot to describe the it in a global frame. Modifications required to the algorithm because of the use of quaternions in the process model are explained. The results of the pose estimation algorithm in relation to the internal sensor data are also presented.

5.1 Virtual Chassis

A snake robot is able to create various shapes and motions with its modules. It is desired that the pose and motion of the snake robot can be intuitively understood by an operator. Wheeled robots are fairly straight forward in that they are typically one rigid body that has an easily definable concept of direction such as 'forward' or 'up'. Defining the frame and direction of a snake robot is more complicated as it has a series of 10 rigid bodies where each module can have a different pose. The overall sense of pose can be obtained by looking at all the modules as a whole; this is defined by Rollison as the virtual chassis [54].

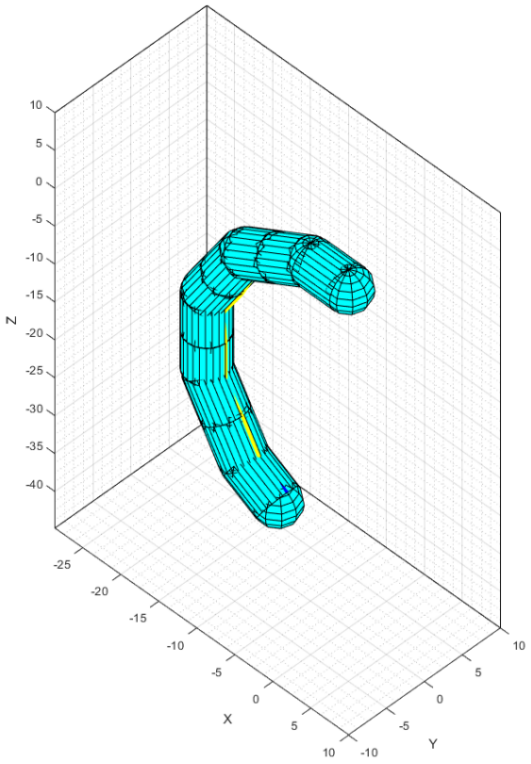
Using a virtual chassis abstracts away from the snakes internal shape and can be used to treat the snake robot as if it were a wheeled robot. The virtual chassis has a body frame whose origin is at the robot's centre of mass with the axes aligned along the principle moments of inertia. The internal shape changes of the robot are separated from the motion of the virtual chassis which allows for the use of simple constant velocity models.

Previous work conducted on the University of Canterbury snake robot by Gilani implemented a virtual chassis for motion trajectory analysis [55]. The snake robot first has to be constructed using a body frame fixed to the first module, the 'head', which is placed at the origin as shown in Figure 5.1a. A standard kinematics model is used to calculate the pose of all the modules with respect to the fixed body frame. A transformation matrix is then found to convert the pose of each module in the fixed frame to that of the virtual chassis shown in Figure 5.1b.

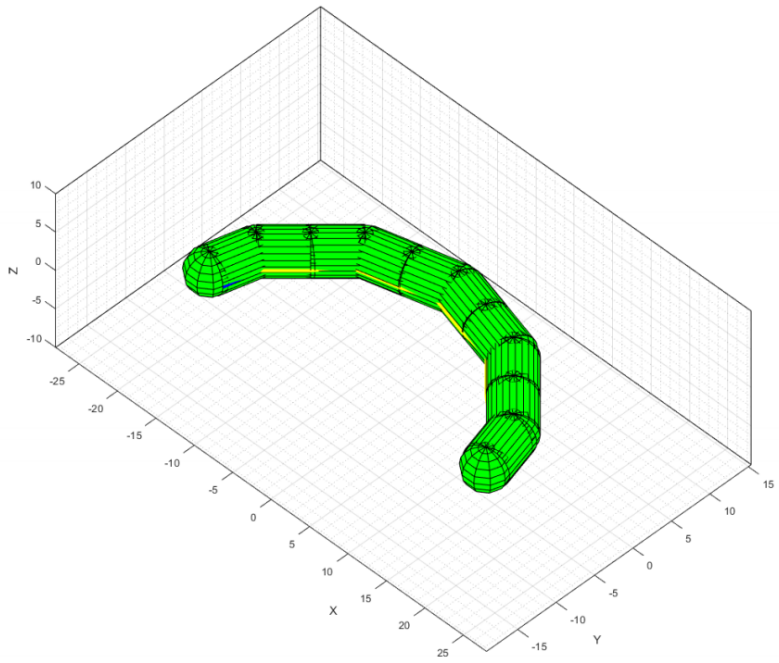
The transformation matrix is calculated by first finding the geometric centre of mass, $\bar{x}$, $\bar{y}$ and $\bar{z}$ of the robot in the fixed-world frame using forward kinematics. A matrix $\boldsymbol{P}$ is constructed containing the position of each module relative to the centre of mass as shown in Equation 5.1 where n equals the number of modules.

$$\boldsymbol{P} = \begin{bmatrix} x_1 - \bar{x_1} & y_1 - \bar{y_1} & z_1 - \bar{z_1} \\ x_2 - \bar{x_2} & y_2 - \bar{y_2} & z_1 - \bar{z_2} \\ \vdots & \vdots & \vdots \\ x_n - \bar{x_n} & y_n - \bar{y_n} & z_n - \bar{z_n} \end{bmatrix} \tag{5.1}$$

To find the rotation matrix so that the principle axis of the fixed-world frame aligns



(a) Frame fixed to the head module.



(b) Virtual chassis frame

Figure 5.1: Comparison between the frame fixed to the head module and the virtual chassis. (retrieved from [1])

with the principle moments of inertia, a singular value decomposition (SVD) of $\boldsymbol{P}$ is conducted as shown in Equation 5.2.

$$\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T = \boldsymbol{P} \tag{5.2}$$

In the decomposition, $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices and $\mathbf{S}$ is a diagonal matrix containing the singular values of $\mathbf{P}$. $\mathbf{V}$ serves as the rotation matrix that describes the desired rotation. Using the SVD, $\mathbf{V}$ is only unique up to a reflection of each set of singular vectors [56]. At the first time step, $\mathbf{V}$ is ambiguous so the third singular vector is set to the cross product of the first and second singular vectors. Across subsequent time steps, the dot products between the current and previous time steps are used to prevent sign flips occurring. The homogeneous transformation matrix can finally be constructed using $\mathbf{V}$ and the centre of mass $\bar{\mathbf{p}}$ as shown in Equation 5.3.

$$\boldsymbol{T} = \begin{bmatrix} \mathbf{V} & \bar{\mathbf{p}} \\ 0 & 0 \end{bmatrix} \tag{5.3}$$

Using the transformation matrix, the pose of each module in the virtual chassis can be found by left multiplying each modules transform by $\mathbf{T}^{-1}$.

5.2 The Unscented Kalman Filter

A UKF was chosen to perform the pose estimation of the snake robot due to the complexity in determining the Jacobians in the EKF. As outlined in Chapter 2, the Square Root Spherically Simplex Unscented Kalman Filter (SR-SSUKF), is a more efficient version of the UKF and has similar performance to the EKF and was therefore implemented for pose estimation.

The Kalman filter operates in two steps: the predict step, and the update step. For the SR-SSUKF, the update step computes the sigma points, predicts the sigma state points and calculates the predicted state and covariance. The sigma points, $\boldsymbol{\mathcal{X}}_{k-1}$, are computed using Equation 5.4 where α determines the spread of the sigma points, $\mathbf{S}$ is the square-root of the state covariance and $\boldsymbol{\mathcal{Z}}$ is the point

selection matrix for the Spherical Simplex Unscented Transform [28].

The sigma points are propagated through the process model, f, to find the sigma state points, as shown in Equation 5.5. The predicted state, $\hat{\boldsymbol{x}}_k^-$, is calculated as shown in Equation 5.6 using the sigma weights $\boldsymbol{W}$ which are precomputed as described by Julier and Eun-Hwan [57][58]. The predicted square-root covariance $\boldsymbol{S}_k^-$ is calculated using a QR decomposition of the error of the sigma state points from the predicted state and the process noise covariance $\boldsymbol{Q}$. A subsequent Cholesky update is necessary as the zeroth weight can be negative [59].

$$\boldsymbol{\mathcal{X}}_{k-1} = \hat{\boldsymbol{x}}_{k-1} + \alpha \mathbf{S}_{k-1} \boldsymbol{\mathcal{Z}} \tag{5.4}$$

$$\boldsymbol{\mathcal{X}}_{k|k-1} = f(\boldsymbol{\mathcal{X}}_{k-1}) \tag{5.5}$$

$$\hat{\boldsymbol{x}}_k^- = \sum_{i=0}^{L+1} W_i \boldsymbol{\mathcal{X}}_{i,k|k-1} \tag{5.6}$$

The update step calculates the sigma measurement points, predicts the sensor measurements and updates the state and state covariance. The sigma state points are propagated through the measurement model, h, to find the sigma measurement points, $\boldsymbol{\mathcal{Y}}_{k|k-1}$, as shown in Equation 5.7. The predicted sensor measurements, $\hat{\boldsymbol{y}}_k^-$, are calculated using the sigma measurement points as shown in Equation 5.8 using the sigma weights. The square-root innovation covariance $\boldsymbol{S}_{\hat{\boldsymbol{y}}k}$ is calculated using a QR decomposition (and subsequent Cholesky update) of the error of the sigma measurement points from the predicted measurements, and the measurement noise covariance $\boldsymbol{R}$ [59].

$$\boldsymbol{\mathcal{Y}}_{k|k-1} = h(\boldsymbol{\mathcal{X}}_{k|k-1}) \tag{5.7}$$

$$\hat{\boldsymbol{y}}_k^- = \sum_{i=0}^{L+1} W_i \boldsymbol{\mathcal{Y}}_{i,k|k-1} \tag{5.8}$$

The Kalamn gain, $\boldsymbol{K}_k$, is found by solving Equation 5.9 using the covariance $\boldsymbol{P}_{\boldsymbol{x}_k \boldsymbol{y}_k}$ and the square-root innovation covariance $\boldsymbol{S}_{\hat{\boldsymbol{y}}k}$. The state is updated using Equation 5.10 with the Kalman gain. The state covariance is finally updated by applying a Cholesky update to $\boldsymbol{S}_k^-$.

$$\boldsymbol{K}_k(\boldsymbol{S}_{\hat{\boldsymbol{y}}k}\boldsymbol{S}_{\hat{\boldsymbol{y}}k}^T) = \boldsymbol{P}_{\boldsymbol{x}_k\boldsymbol{y}_k} \tag{5.9}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k^-) \tag{5.10}$$

5.2.1 State Vectors

The state vector contains the parameters used in the process model to predict the pose of the snake robot. Two general model approaches that are used for the snake robot are: using the gait parameters used to generate the gait pattern or using the joint angles directly.

The first method is to use the prior knowledge of the gait pattern being used meaning that the gait parameters can be used to model the pose of the snake robot. Using gait parameters, the length of the state vector can be reduced and be independent from the number of modules. If an error term $\mathbf{e}$ is used in the state vector to track individual modules deviation from the gait pattern, then more accurate state estimation is possible. Adding the error term does make the state vector length dependent on the number of modules, however the advantages of tracking the snake robot when modules fail make including it more useful than the minimising of the length of the state vector if it is not included. The gait parameters used to estimate the pose of the snake robot can be simplified for each gait pattern by reducing the number of gait parameters required which gives better stability to the model.

The second method using the joint angles is a more intuitive model, and requires no prior knowledge of the movement of the snake robot. The joint angle method does not depend on the robot performing a specific gait so could be used as a generic model. Using a third order model of the joint angles allows them to be accurately tracked. The disadvantage of the joint angle approach is the that the state vectors length is dependent on the number of modules in the snake robot.

Rolling Gait

The rolling gait propagates two sin waves through the yaw and pitch modules offset by the phase ψ_{py}. The offset is assumed to be constant and set to 90 degrees ($\frac{\pi}{2}$), the same offset as between the snake modules. The rolling gait was simplified from the sinusoid function generator to Equation 5.11. The gait parameter ω is combined with the time t as shown in Equation 5.12. The state vector was constructed to contain the gait parameters used in Equation 5.11 and their second order component shown in Equation 5.13.

$$\varphi(t)_i = \begin{cases} -Asin(V + \psi_{py}) & \text{if } i = \text{even} \\ Asin(V) & \text{if } i = \text{odd} \end{cases} \tag{5.11}$$

$$V = \omega t \tag{5.12}$$

$$\mathbf{x_k} = \begin{bmatrix} A_k & \dot{A}_k & V_k & \dot{V}_k & \mathbf{e}_k & \mathbf{q}_k & \boldsymbol{\omega}_k \end{bmatrix} \tag{5.13}$$

Where $\mathbf{e} = [e_1 ... e_n]$ is the error of each joint angle for n modules , $\mathbf{q} = [q_1 q_2 q_3 q_4]$ is the quaternion of the virtual chassis in the world frame, and $\boldsymbol{\omega} = [\omega_x \omega_y \omega_z]$ is the angular velocity of the virtual chassis. The length of the state vector for the rolling model is 20.

Linear Progression Gait

The linear progress gait propagates a single sin wave through the pitch modules where each module is out of phase by ψ. The yaw modules are set to 0 degrees in the case of going straight or used for the turning gait by setting the angle O_y to the desired turn rate. The linear progression gait was simplified from the sinusoid function generator to Equation 5.14. The state vector in Equation 5.15 was constructed using the parameters in Equation 5.14, and has a length of 24.

$$\varphi(t)_i = \begin{cases} O_y & \text{if } i = \text{even} \\ Asin(V + \psi n) & \text{if } i = \text{odd},\ n \in \{1...M_p\} \end{cases} \tag{5.14}$$

$$\mathbf{x_k} = \begin{bmatrix} A_k\ \dot{A}_k\ V_k\ \dot{V}_k\ \psi_k\ \dot{\psi}_k\ O_{yk}\ \dot{O}_{yk}\ \mathbf{e}_k\ \mathbf{q}_k\ \boldsymbol{\omega}_k \end{bmatrix} \tag{5.15}$$

Rotating Gait

The rotating gait is based on the sidewinding gait without any forward motion. A sin wave is propagated through the yaw and pitch models each with a different amplitude. Each module is out of phase by either ψ_y or ψ_p to create the rotating motion. The rotating gait can be simplified from the sinusoid function generator to Equation 5.16. The state vector in Equation 5.17 was constructed using the parameters in Equation 5.16, and has a length of 26.

$$\varphi(t)_i = \begin{cases} -A_y sin(V + \psi_y n) & \text{if } i = \text{even},\ n \in \{1...M_y\} \\ A_p sin(V + \psi_p n) & \text{if } i = \text{odd},\ n \in \{1...M_p\} \end{cases} \tag{5.16}$$

$$\mathbf{x_k} = \begin{bmatrix} A_{yk}\ \dot{A}_{yk}\ A_{pk}\ \dot{A}_{pk}\ V_k\ \dot{V}_k\ \psi_{yk}\ \dot{\psi}_{yk}\ \psi_{pk}\ \dot{\psi}_{pk}\ \mathbf{e}_k\ \mathbf{q}_k\ \boldsymbol{\omega}_k \end{bmatrix} \tag{5.17}$$

Joint Angles Model

The joint angle model does not use prior knowledge of the gait parameter used and instead uses a third order system for modelling the joint angles as shown in the state vector in Equation 5.18, which has a length of 34.

$$\mathbf{x_k} = \begin{bmatrix} \theta_k\ \dot{\theta}_k\ \ddot{\theta}_k\ \mathbf{q}_k\ \boldsymbol{\omega}_k \end{bmatrix} \tag{5.18}$$

5.2.2 Process Model

For each of the gait parameters in the state vectors, a second order model is used. The following equations show the second order model used for the rolling gait:

$$\hat{A}_k = A_{k-1} + \dot{A}_{k-1}dt \tag{5.19}$$

$$\hat{\dot{A}}_k = \dot{A}_{k-1} \tag{5.20}$$

$$\hat{V}_k = V_{k-1} + \dot{V}_{k-1}dt \tag{5.21}$$

$$\hat{\dot{V}}_k = \dot{V}_{k-1} \tag{5.22}$$

The other gait parameters for each of the state vectors used shares the same form of process model. The error parameter is common to all of the gait parameter models and is predicted using Equation 5.23 for each i^{th} module. λ is a dampening coefficient that set to 0.95 as suggested by Rollison to model the effect of the proportional controllers running in each motor [31].

$$\hat{\mathbf{e}}^i_k = \lambda \mathbf{e}^i_{k-1} \tag{5.23}$$

The quaternion and angular velocity models are the same for every choice of state vector. The quaternion model in Equation 5.24 uses a discrete-time update developed by Van Der Merwe as shown in Equation 5.25 [27].The angular velocity is assumed to be constant across time steps as shown in Equation 5.26.

$$\hat{\mathbf{q}}_k = \exp(-1/2\mathbf{\Psi}dt)\mathbf{q}_{k-1} \tag{5.24}$$

$$\mathbf{\Psi} = \begin{bmatrix} 0 & \omega_{x_{k-1}} & \omega_{y_{k-1}} & \omega_{z_{k-1}} \\ -\omega_{x_{k-1}} & 0 & -\omega_{z_{k-1}} & \omega_{y_{k-1}} \\ -\omega_{y_{k-1}} & \omega_{z_{k-1}} & 0 & -\omega_{x_{k-1}} \\ -\omega_{z_{k-1}} & -\omega_{y_{k-1}} & \omega_{z_{k-1}} & 0 \end{bmatrix} \tag{5.25}$$

$$\hat{\boldsymbol{\omega}}_k = \boldsymbol{\omega}_{k-1} \tag{5.26}$$

5.2.3 Measurement Vector

The same measurement data is collected regardless of the model used; the joint angle of each module except the head module, the roll of each model, the pitch of each module, and the yaw from the head module. The measurement vector shown in Equation 5.27 can therefore be used for all of the models, which has a dimension of $3n$, where n is the number of modules.

$$\mathbf{z}_k = \lfloor \boldsymbol{\Phi}_k \ \boldsymbol{\alpha}_k \ \boldsymbol{\beta}_k \ \gamma_k \rfloor \tag{5.27}$$

In Equation 5.27, $\boldsymbol{\Phi}$ is the vector of joint angles, $\boldsymbol{\alpha}$ is the vector of roll angles, $\boldsymbol{\beta}$ is the vector of pitch angles and γ is the yaw of the head module.

5.2.4 Measurement Model

The measurement model for all of the process models is similar with just the joint angle prediction being different. For the models using the simplified gait parameters, the gait pattern equations are used to find the joint angles. The joint angle model naturally uses the joint angles predicted directly.

The joint angles used a standard forward kinematics model for the virtual chassis as outlined in Section 5.1. Each modules pose is found by aligning the module to the virtual chassis then multiplying it by the quaternion as shown in Equation 5.28, where $\hat{\mathbf{R}}_\mathbf{k}$ is the rotation matrix representation of the estimate of the quaternion $\hat{\mathbf{q}}_\mathbf{k}$

$$\hat{\mathbf{M}}^i_{qk} = \hat{\mathbf{R}}_k \hat{\mathbf{M}}^i_k \tag{5.28}$$

The rotation matrix, $\hat{\mathbf{M}}^i_{qk}$ takes the form as shown in Equation 5.29, from which the roll, pitch and yaw angles can be extracted [60]. To account for each of the yaw modules being offset by 90 degrees, the roll measurement is also offset by 90 degrees.

$$\mathbf{M} = \mathbf{M}_z(\gamma)\mathbf{M}_y(\beta)\mathbf{M}_x(\alpha) \tag{5.29}$$

5.2.5 Difficulties Using Quaternions in the SR-SSUKF

Using quaternions in the process model causes problems implementing a normal SR-SSUKF as the quaternion is a unit norm, which is not in the vector space, and the parameters are not independent. The SR-SSUKF was adapted to handle quaternions using the method outline by Kraft, where a three dimensional rotation vector is used to generate the sigma points [61][62]. This approach reduces the dimensionality of the state covariance and process noise covariance by one, so a transform is used to convert the quaternions between representations.

The weighted mean used to find the predicted state in Equation 5.6 contains quaternions which are not in a vector space and therefore does not return correct results. The method to solve this problem presented by Kraft uses an iterative gradient descent algorithm [61]. Starting at an arbitrary orientation, the mean orientation $\bar{\boldsymbol{q}}$ is computed for each step t. The error vector, $\vec{\boldsymbol{e}}$, is computed for each sigma point using Equation 5.30 where $\boldsymbol{e}_i$ is the quaternion representation of $\vec{\boldsymbol{e}}$. The estimated weighted mean is then found as shown in Equation 5.31.

$$\boldsymbol{e}_i = \boldsymbol{q}_i \bar{\boldsymbol{q}}_t^{-1} \tag{5.30}$$

$$\vec{\boldsymbol{e}} = \sum_{i=0}^{L+1} W_i \vec{\boldsymbol{e}}_i \tag{5.31}$$

The corresponding quaternion of $\vec{\boldsymbol{e}}$ is used to calculated a better estimate of $\bar{\boldsymbol{q}}$ as shown in Equation 5.32. $\vec{\boldsymbol{e}}$ tends towards zero as the calculated mean approaches the real mean, so it is used to stop the iterative process once the desired precision is achieved. The set $\{\vec{\boldsymbol{e}}_i\}$ of the final iteration in Equation 5.31 is used in the calculation of the covariances as described by Kraft [61].

$$\bar{\boldsymbol{q}}_{t+1} = \boldsymbol{e}\bar{\boldsymbol{q}}_t \tag{5.32}$$

5.2.6 Tuning the Kalman Filter

The generation of the sigma points is scaled by three scaling parameters, α, κ and β as described by Julier and Eun-Hwan [57][58]. α sets the spread of the sigma points and is normally set to $1e^{-4} \leq \alpha \leq 1$; κ is a second scaling parameter normally set to zero; and β incorporates prior knowledge of the distribution of the state which, for a Gaussian distribution, $\beta = 2$ is optimal.

The state vector using the gait parameters models contain elements that are of different magnitudes and lie in different spaces like V which is in the space of $\pm\pi$. The generation of the sigma points is therefore scaled individually for each parameter to keep the generated sigma points within the different parameter domains. Using the amplitude parameter A as a baseline, the scaling factor α was set to 0.5 based on the expected spread of sigma points. A scaling factor for each other parameter was experimentally determined based on the desired spread of the parameter. Table 5.1 shows the scaling values for the parameters used in the linear progression gait. The scaling factors for the other models are shown in Appendix C.

Table 5.1: Scaling factors for the linear progression gait.

Parameter	**Scaling factor**
A	1
$\dot{A}$	1
V	0.05
$\dot{V}$	0.5
ψ	0.05
$\dot{\psi}$	0.5
O_y	1
$\dot{O}_y$	1
$\boldsymbol{e}$	1
$\boldsymbol{q}$	0.2
$\boldsymbol{\omega}$	0.1

The SR-SSUKF is initialised based on the selected process model. The state vector parameters are set using the gait parameters or if the joint angles model is selected, the state vector parameters are initialised to zero. The quaternion is a unit norm so the scalar part of the quaternion is set to one. The state covariance matrix is initialised as an identity matrix.

The process noise covariance matrix $\boldsymbol{Q}$ and the measurement noise covariance

matrix $\boldsymbol{R}$ are used to tune the confidence of the states and measurements with respect to each other. The process noise covariance was determined experimentally and was set to an identity matrix. An estimate of twice the resolution of the encoders on the Herkulex DRS-0101 smart servos was used giving a variance of 0.4225 degrees. The roll, pitch and yaw variances were experimentally determined and set to 2 degrees. The covariance matrix R is shown in Equation 5.33 where the first 9 columns are the servo motor measurements and the last 21 are the roll, pitch and yaw measurements.

$$R = \begin{bmatrix} 0.4225 & & & & & & \\ & \ddots & & & & & \\ & & 0.4225 & & & & \\ & & & 2 & & & \\ & & & & \ddots & & \\ & & & & & 2 & \end{bmatrix} \tag{5.33}$$

5.2.7 Failed Sensor Readings

Due to the snake robot having multiple sensors which return data on the same data bus, failure to retrieve all sensor data is unavoidable. To accommodate failed sensor readings, the measurement reading is set to the predicted measurement, and the measurements corresponding element in the measurement noise covariance $\boldsymbol{R}$ is set to a relatively high value of 10^6, effectively ignoring the measurement in the update step.

5.3 Internal Pose Estimation Results

The accuracy of the SR-SSUKF tracking the sensor data, was tested in the development room. The test was set up by selecting the desired gait and running it across the test area as shown in Figure 5.2. The sensor data and predicted data at each time step was recorded through a log file.

The motor angles for modules 1 and 2 using the rolling gait are shown in Figure 5.3 where the predicted angle is closely tracking the motor encoders. Figure 5.4

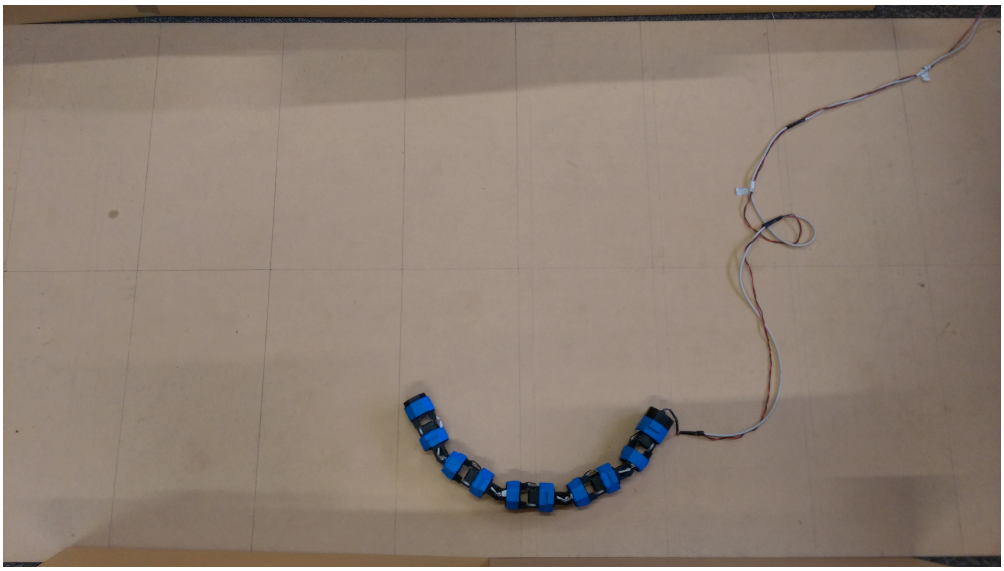


Figure 5.2: Setup of the test area for the internal tests.

shows the absolute error between the motor encoders and the predicted angle which shows a consistently small error, except as the motors start and stop. The average motor error for the rolling gait was 0.30 degrees. The other gaits were tested and had a similar prediction accuracy.

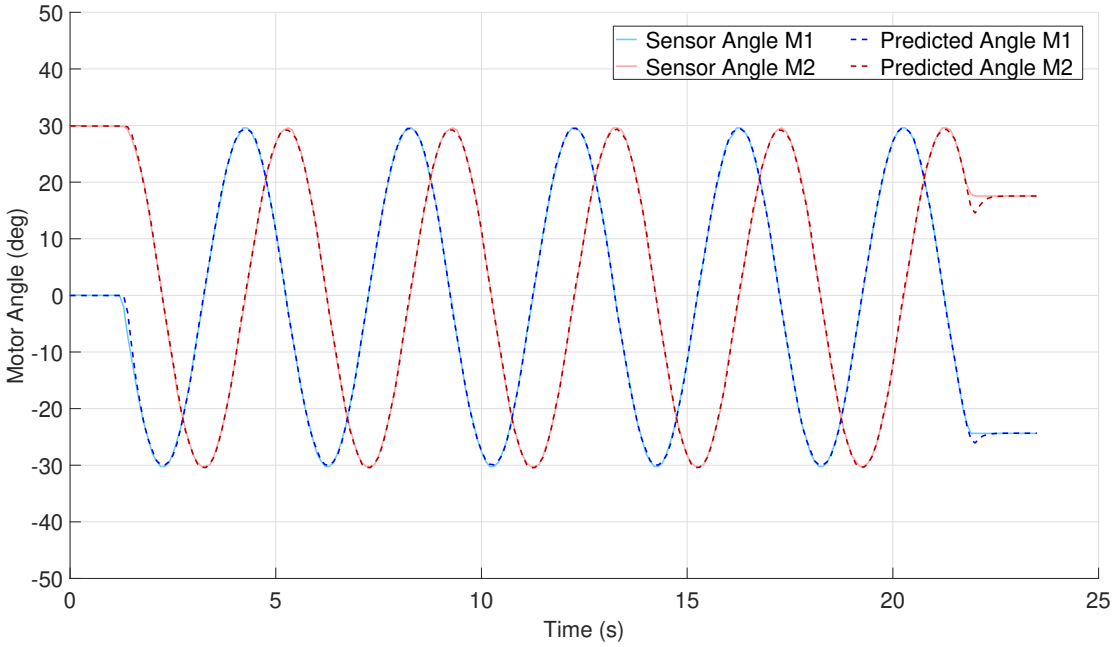


Figure 5.3: Motor angles of modules 1 and 2 while performing the rolling gait.

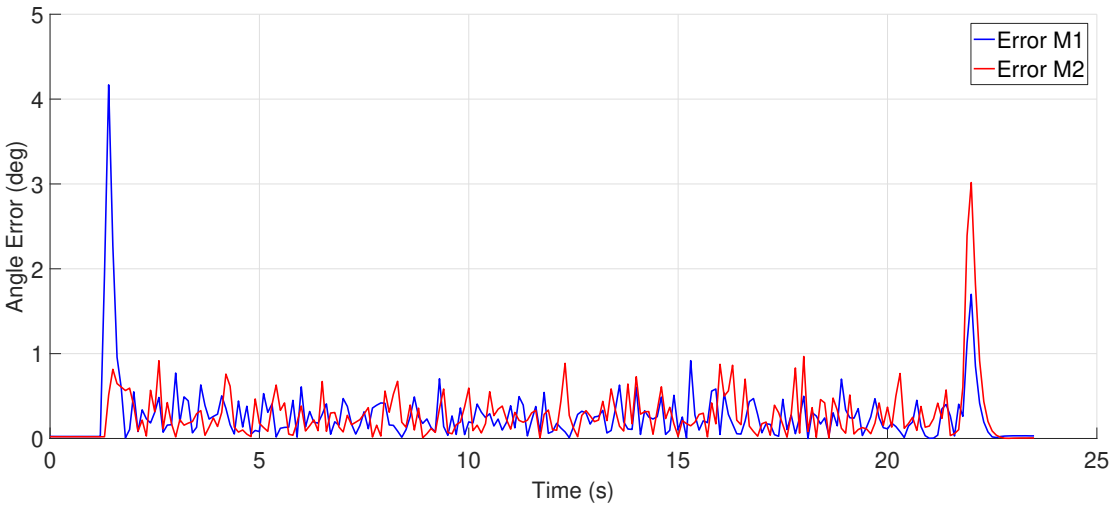
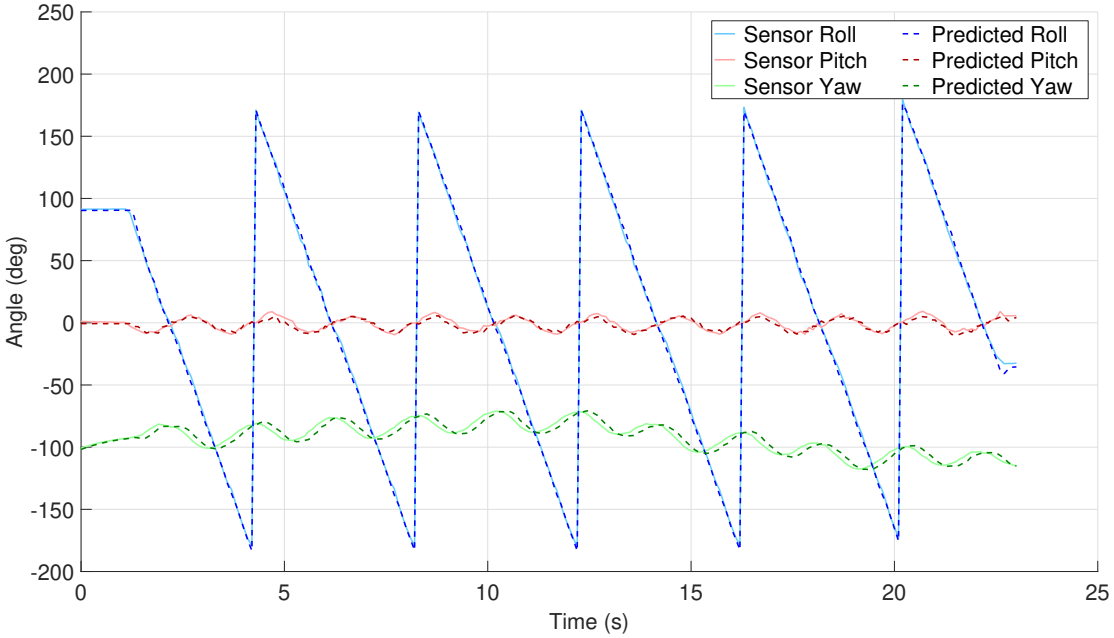
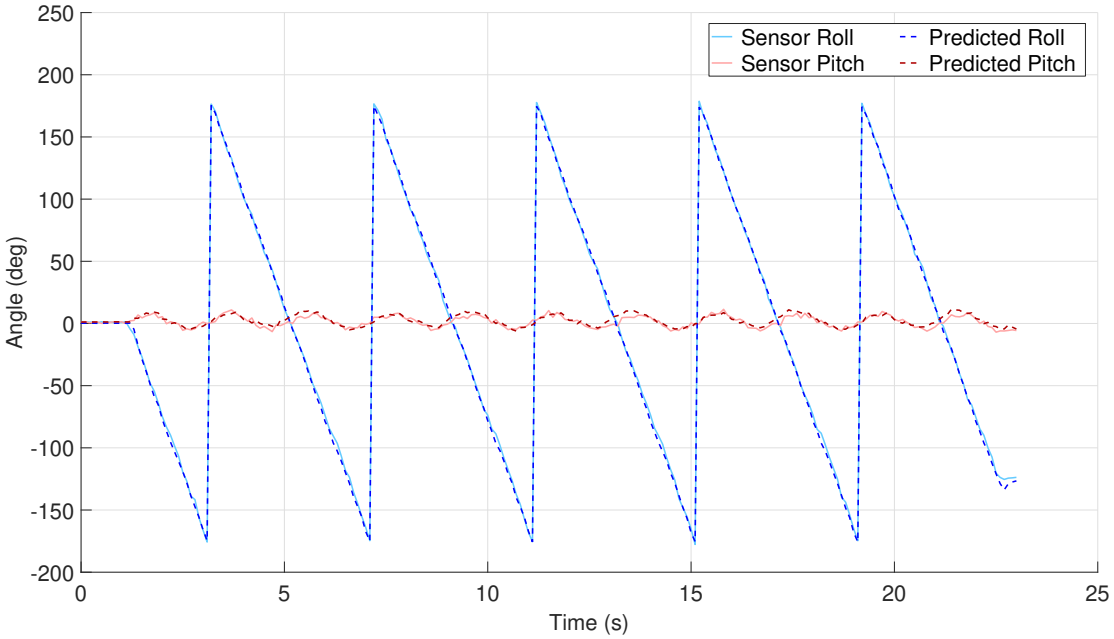


Figure 5.4: Motor angle errors of modules 1 and 2 while performing the rolling gait.

The tracking of each of the modules orientation shows how well the SR-SSUKF is predicting the orientation of the snake robot as a whole. The tracking of the Euler Angles for the rolling gait in Module 0 and 9 can be seen in Figure D.5. As only the head module, module 0, outputs yaw data from the IMU, only module 0 shows the yaw prediction. The error graph for modules 0 and 9 showing the absolute error between the sensor data and the predicted data can be seen in Figure D.12.

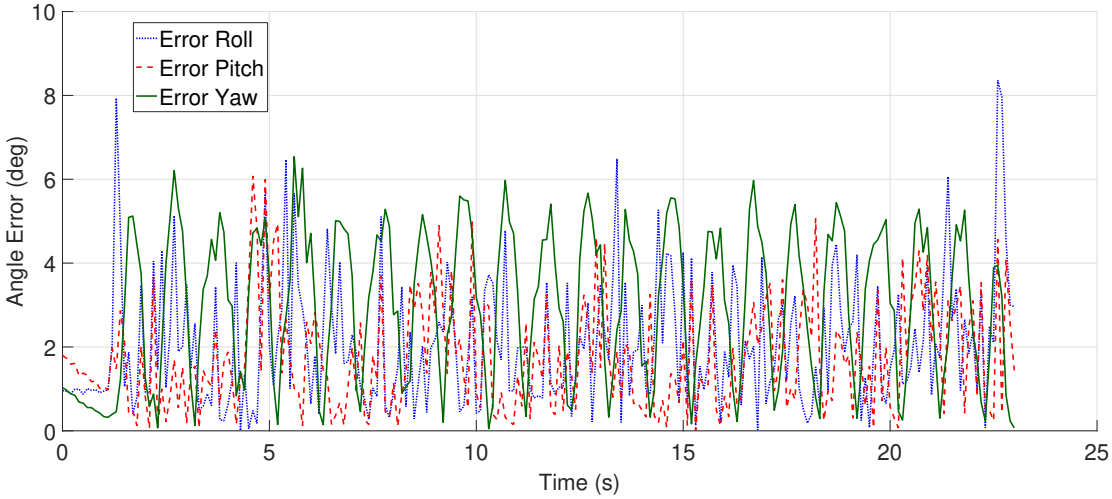


(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the rolling gait using the gait parameters model.

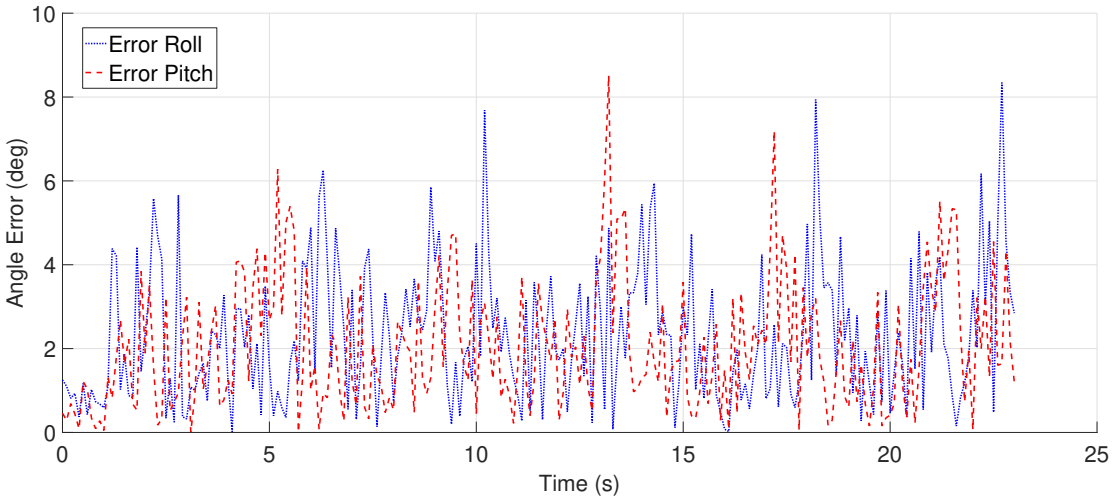


(b) Comparison between the predicted Euler angles and the sensor data for module 9 while performing the rolling gait using the gait parameters model.

Figure 5.5: Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the gait parameters model.



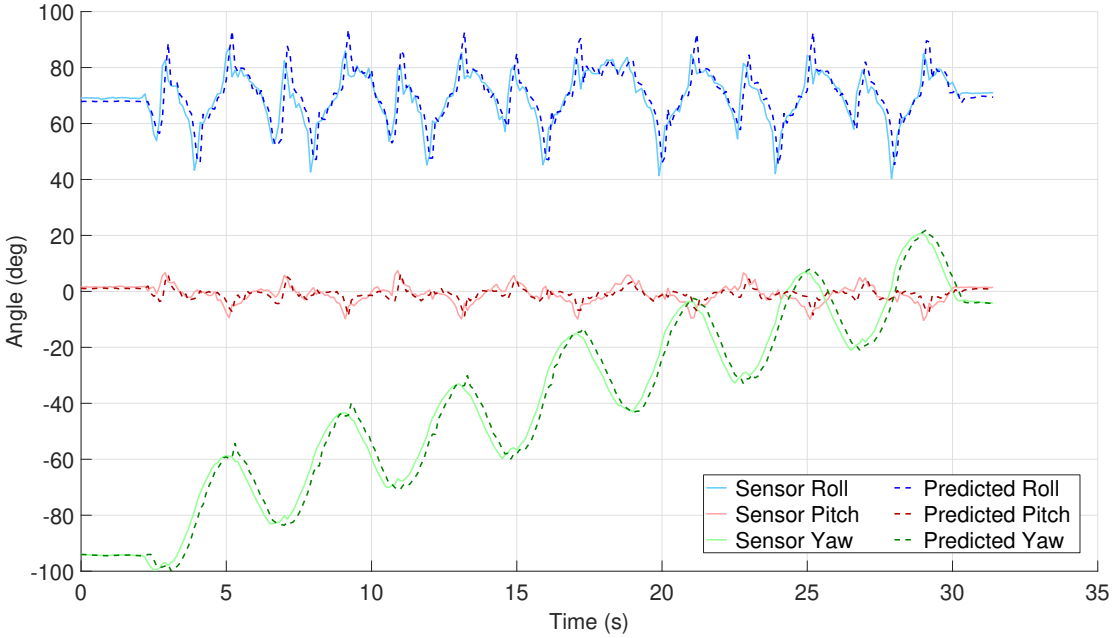
(a) Prediction error of module 0 using the rolling gait with the gait parameters model.



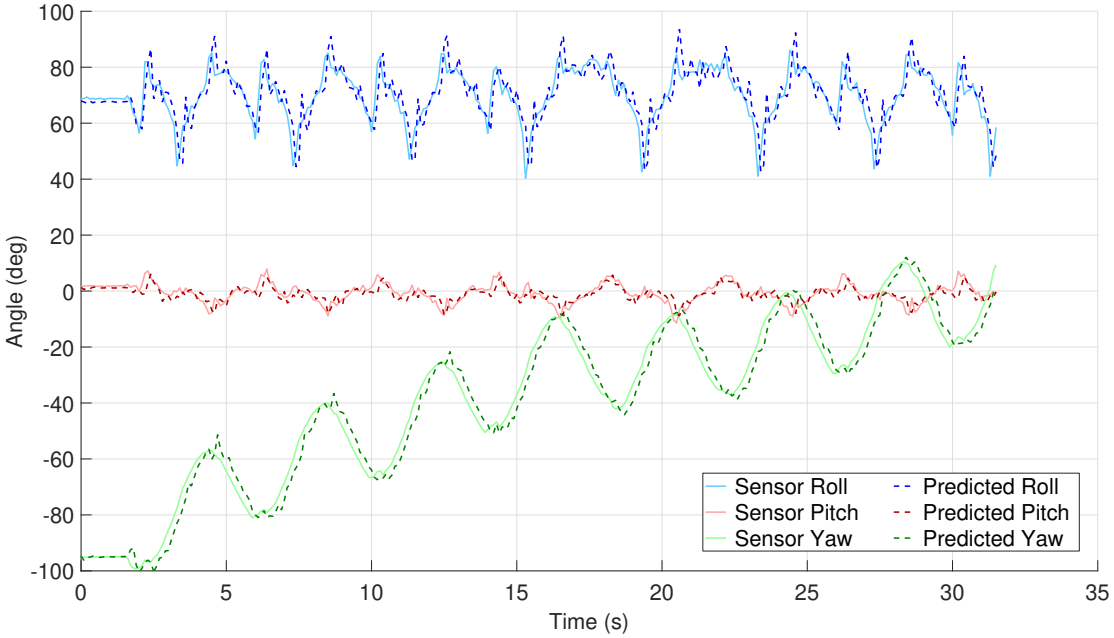
(b) Prediction error of module 9 using the rolling gait with the gait parameters model.

Figure 5.6: Predicted error of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the gait parameters model.

The rotating gait is the most complex gait pattern tested using the pose estimation algorithm so was used to compare the gait parameters model to the joint angle model. Figure 5.7a shows the gait pattern model and Figure 5.7b shows the joint angles model. It can be seen that they both have similar prediction accuracy.



(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the rotating gait using the gait parameters model.



(b) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the rotating gait using the joint angles model.

Figure 5.7: Comparison between the SR-SSUKF prediction using the gait parameters model and the joint angle model while using the rotating gait.

The results of the other gait patterns tested can be seen in Appendix D which includes both the prediction graphs and error graphs for Module 0 and 9. The mean error of the Euler angles for each of the gaits tested is shown in Table 5.2. It can be seen that the linear progression which is the simplest gait pattern has the smallest mean error while the rotating gait was more complex and had a greater mean error. The gait parameter model and the joint angle model performed similarly across all of the motion gaits with the gait parameter model having a slightly lower mean error.

Table 5.2: Module 0 mean Euler angle prediction error.

Gait	**Roll (Degrees)**	**Pitch (Degrees)**	**Yaw (Degrees)**
Linear Progression	0.94	1.42	0.28
Turning	1.58	1.10	0.42
Rolling	2.07	1.72	3.00
Rotating	3.78	1.88	3.35
Angles Linear Progression	0.99	1.57	0.30
Angles Turning	2.49	1.16	0.74
Angles Rolling	2.43	1.75	2.93
Angles Rotating	4.30	1.82	3.57

5.4 Summary

To make intelligent control decisions for the snake robot, feedback is required for closed loop control. The sensor data collected from the snake robot modules was used to develop a pose estimation algorithm to predict the orientation of each module and of the snake robot as a whole.

The choice of body frame for the snake robot is important to make the operation of the snake robot intuitive. Using a virtual chassis abstracts away from the snake robots internal shape to allow it to be treated as if it were a wheeled robot. The virtual chassis has a body frame whose origin lies at the centre of mass with the axes aligned along the principle moments of inertia. The virtual chassis is constructed with a body frame fixed to the head module, which is placed at the origin. A standard kinematics model is used to calculate the pose of all the modules with respect to the fixed body frame. A transformation matrix is then found to convert the pose of each module into the virtual chassis frame.

The SR-SSUKF was selected to perform the pose estimation of the snake robot

with modifications to handle the quaternions in the state vector. The sinusoid function generator was simplified for each of the gaits to create a simple state vector for each gait. An additional model directly using the joint angles was included to test the accuracy between the two different approaches. A process model for each of the state vectors was created using a simple second order model for each of the gait parameters and a third order model for the joint angles. The measurement vector for all of the models is the same since the same sensor data is collected. The only difference in the measurement model is the equation used to generate the joint angles. The joint angles are used in the forward kinematics model to find the pose of each module which is transformed into the virtual chassis frame.

The pose estimation algorithm was tuned with the scaling factor set to 0.5 based on the amplitude parameter A. A scaling factor for each of the other state parameters were determined experimentally. The process covariance was also determined experimentally was set to an identity matrix. The measurement covariance was initialised for the motors based an estimate of twice the encoder resolution and was set to 0.4225 degrees. The measurement covariance for the roll, pitch and yaw of the modules was determined experimentally and set to 2 degrees. The state vectors where initialised using the gait parameters or set to zero in the case of the joint angle model.

The pose estimation algorithm was tested internally against the sensor data and shows that the motor prediction is accurate apart from when the robot starts or stops. The mean prediction error for the linear progression and turning gaits are smaller than for the rolling and rotating gaits, which is to be expected as they are less complex gait patterns. The joint angles model had similar performance to the gait parameters model though the latter performed slightly better across all of the motion gaits tested.

Chapter 6

Software Design of Pose Estimation Algorithm

The snake robot is operated using the Robot Operation System (ROS) - Indigo Igloo which uses a collection of nodes that each performs a distinct task. The pose estimation algorithm was developed by building on the existing ROS software by updating the original nodes and creating new, dedicated nodes.

This chapter outlines the software used to implement the pose estimation algorithm presented in Chapter 5. The collection of the sensor data is controlled by the Teensy LCs installed on each of the snake robot modules connected to the Teensy 3.2 via a RS485 data bus. The Teensy 3.2 retrieves the sensor data from the snake robot and sends it to the PC to be used by the pose estimation algorithm ROS node.

6.1 The Robot Operating System

ROS is designed to have multiple concurrently running nodes each with their own specific task ensuring the system remains highly modular. With the implementation of the pose estimation algorithm, ROS uses five nodes:

- Gait Generation
- Graphical User Interface (GUI)

- Serial
- Snake Timing
- Snake UKF

The first three nodes perform the same functions as in the previous prototype system written by Gilani though with some modifications [1]. The snake timing node uses a consistent loop rate of 10 Hz to request the sensor data from the serial node. As described in Chapter 4 the sensor request starts the pose estimation loop, as a consistent sampling rate for the pose estimation algorithm is desired. The UKF node performs the pose estimation of the snake robot using a UKF.

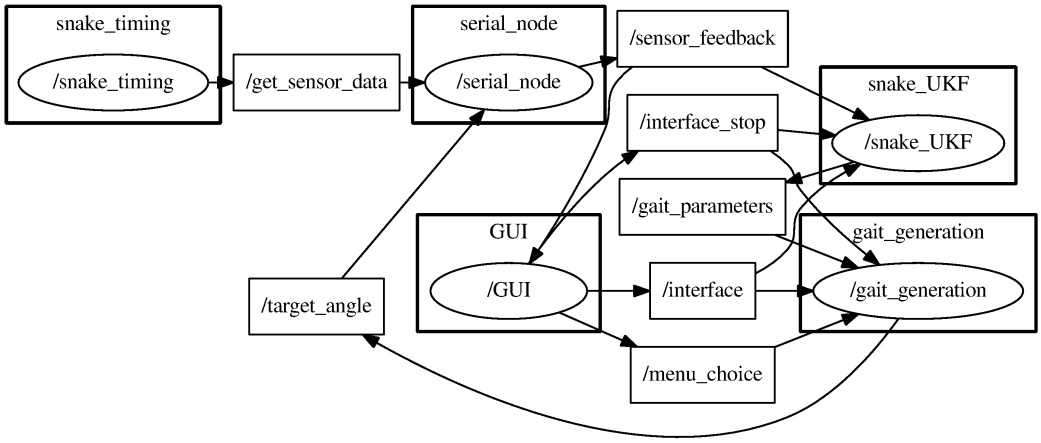


Figure 6.1: ROS graph of the nodes interactions where the ellipses are the nodes, the rectangles are the topics and the arrows are the subscriptions to those topics.

The nodes and the topics that they use to communicate can be seen in Figure 6.1. The *"menu_choice"*, *"interface"* and *"interface_stop"* topics are all published by the GUI node: the *"menu_choice"* topic sets the mode between directly setting motor angles or using a predefined gait pattern; the *"interface"* topic sends the gait parameters for the selected gait or the motor angles if they are being directly commanded; and the *"interface_stop"* topic sends a boolean to start or stop the locomotion of the snake robot.

The gait generation node publishes the *"target_angle"* topic with the generated target motor angles. The snake timing node publishes the *"get_sensor_data"* topic to request the sensor data. The serial node publishes the *"sensor_feedback"* topic which is a custom message type containing the sensor data from all of the snake

modules. The snake UKF node publishes the *"gait_parameters"* topic containing the predicted gait parameters.

The transfer of data between the ROS system running on the computer and the software running on the snake robot can be seen in Figure 6.2.

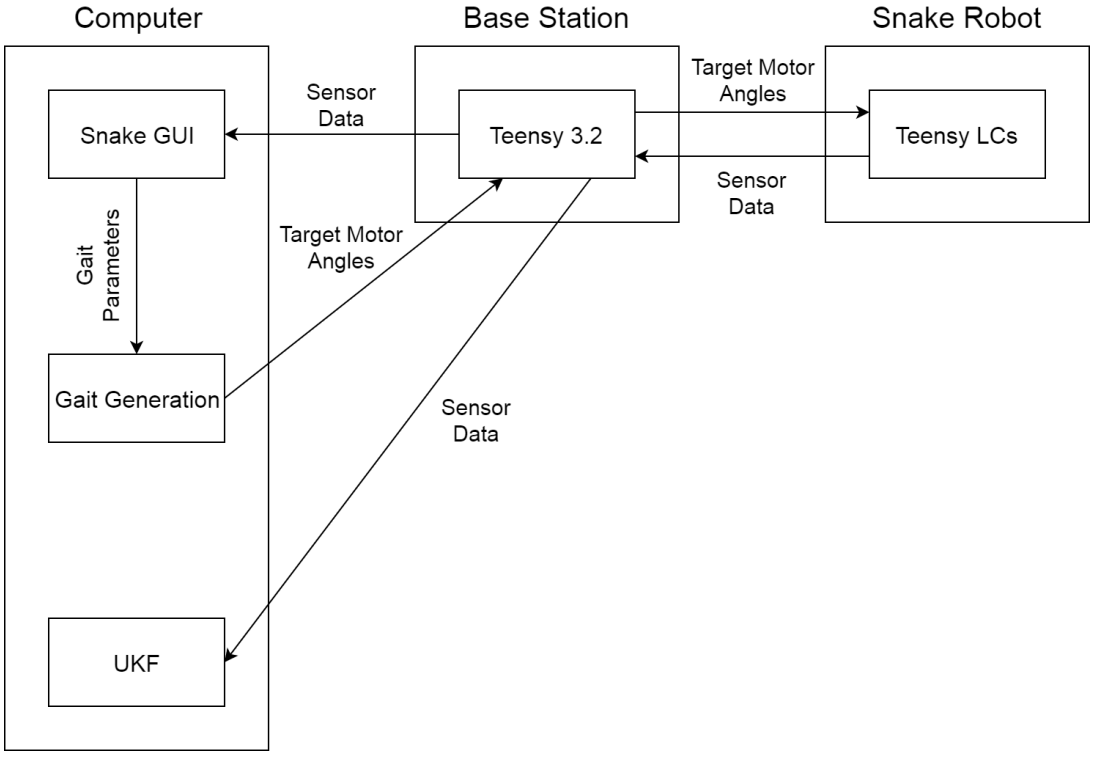


Figure 6.2: Transfer of data between the computer and the snake robot.

6.2 Gait Generation Node

The gait generation node was written by Gilani to calculate the motor angles required to produce the desired gait pattern [1]. The gait generation node subscribes to the *"menu_choice"*, *"interface"* and *"interface_stop"* topics published by the GUI node and the *"gait_parameters"* topic published by the snake UKF node.

When each of the topics publishes a message, the respective callback function shown in Figure 6.3 is called. The *MenuChoice* callback function sets the state of the node between directly using motor angles or calculating motor angles from the received gait parameters. The *Interface* callback receives data from the GUI, and if the motor angles are sent, the received angles are converted into an integer

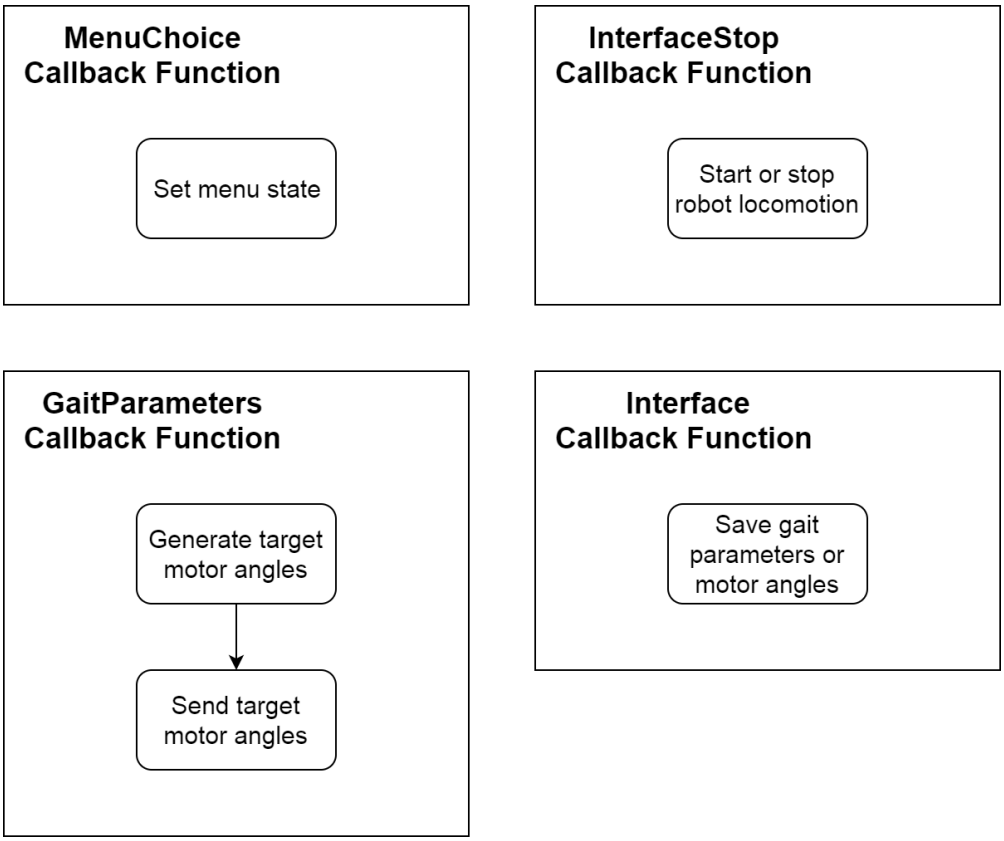


Figure 6.3: Callback functions used by the gait generation ROS node.

between 0 and 1023 expected by the motor where 512 is equivalent to 0 degrees. If gait parameters are sent, they are used in the generic sinusoid function generator in Equation 2.1 to generate the motor angles which are then converted to an integer for the motors. If the snake robot is moving when gait parameters are received, then the program waits until the current gait pattern cycle is completed before using the new gait parameters, which prevents the snake being in the wrong orientation while changing gait patterns. The *InterfaceStop* callback controls the snake starting and stopping motion by setting the timestep used to 0 or 1. The *GaitParameters* callback triggers the next set of target angles to be published without using the received gait parameters.

The converted motor angles are publish on the *"target_angle"* topic which is subscribed to by the serial node running on the Teensy 3.2. The message structure of the *"target_angle"* topic was updated to use an array containing all of the motor angles is used instead of an individual variable for each. Using an array means the message structure is the same regardless of the number of modules.

6.3 Base Station Software Design

The Teensy 3.2 runs the serial node to connect the computer to the snake robot. The tasks performed by the Teensy 3.2 are to send motor angles and retrieve sensor data from the Teensy LC each of the modules. The serial node written by Gilani to test the gait patterns during development sent the motor angles directly to the motors through the daisy chained TTL data bus [1]. With the hardware changes outlined in Chapter 4, the motor angles are instead sent to the Teensy LCs in each module.

To send and receive data from each of the modules, the RS485 standard was selected for its high speed, long distance capabilities with the Teensy 3.2 being the master device. RS485 is a hardware standard, so software is required to drive the data bus.

A RS485 library for Arduino is utilised written by Gammon (2012, Version 1.0), to send and receive data packets between 1 and 255 bytes [63]. The protocol uses an error checking system to ensure that data arrives correctly and intact. The message begins with a start of text (STX) character and finishes with an end of text (ETX) character. Each byte, apart from the STX and ETX characters, is sent in a double/inverted format where the first and last four bits are both sent twice, once normally and once inverted. As one byte of data is sent as two bytes which are symmetric, there are only 15 possibilities, 0F, 1E, 2D and so on. If one of the bytes received isn't one of the 15 expected bytes, them an error must have occurred. At the end of the message, a checksum is sent to ensure that the whole message has been received intact.

The fastest baud rate that the Teensy LC can reliably use is 1000000 baud, therefore this was the chosen baud rate of the RS485 data bus. To communicate with each snake module using the master/slave design, each device was assigned a unique address corresponding to the module it is located in, while the Teensy 3.2 was given the address of 127. The device address is stored in the permanent EEPROM of each device so the same software can be used. A message structure is used to send and receive data across the RS485 data bus which includes the sender address, the receiver address, the purpose of the message and a data buffer if required. To send a message to all devices instead of an individual device, the sender address is set to 200. Table 6.1 shows the control flags used to determine the purpose of the message. The save request and sensor request do not have data

buffers while the send motor angles and send sensor data messages buffers are as short as possible to reduce the sending time.

Table 6.1: Control flags for message structure.

Purpose Definition	**Flag Number**	**Explanation**
SEND_MOTOR_ANGLES	1	Flag for sending motor angles
SAVE_REQUEST	2	Flag for sending sensor save request
SENSOR_REQUEST	3	Flag for requesting sensor data
SEND_SENSOR_DATA	4	Flag for sending sensor data to the base station

The Teensy 3.2 serial node subscribes to two topics, *"target_angle"* and *"get_sensor_data"*, which trigger the callback functions seen in Figure 6.4. When a message is received on the *"target_angle"* topic, a message containing the motor angles is sent to all of the snake modules to ensure that the motor commands are received at the same time. When a message is received on the *"get_sensor_data"* topic, the sensor collection process shown is started.

To ensure that all the sensor data is collected at the same time, the Teensy 3.2 sends a save request at the start of the process to all of the modules. The save request is interpreted by the Teensy LC devices to save the current sensor data and prepare it for sending. After the save request is sent, a timeout of 6 ms is used to allow the first module time to prepare the sensor data.

The data is then requested in sequential order from device 0, which is the head module, to device 9 which is the tail module. To allow time for the device to respond to the request, a timeout definition is set to 1 ms which is the maximum allowable response time based on the response time of the Teensy LC in Section 6.4.

During the timeout period, the Teensy 3.2 repeatedly polls the serial port to check if a message has been received. If the device responds correctly to the data request, the sensor data from that device is stored in a ROS message. If the device does not respond before the timeout, the sensor data is set 200 and stored in the ROS message to indicated the failed sensor reading. After each request the request address is incremented by one. Once all of the modules have responded, the ROS message containing all of the sensor data is published to the topic *"sensor_feedback"*.

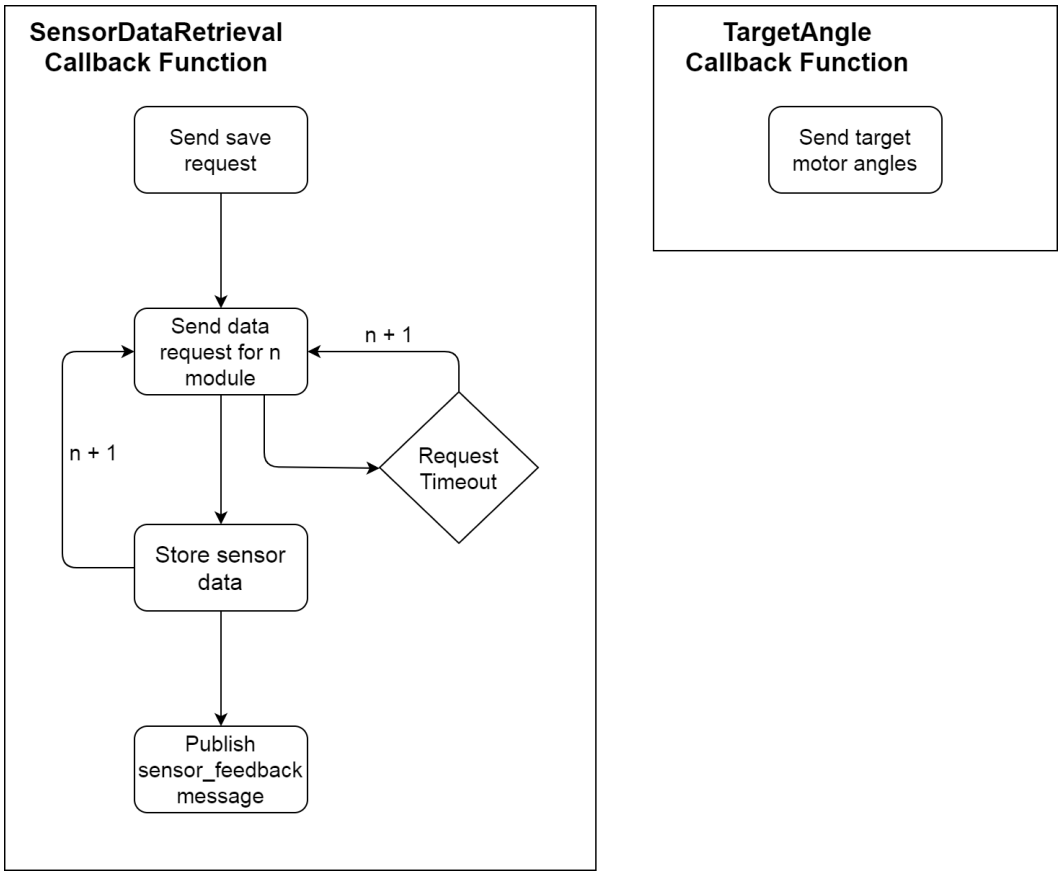


Figure 6.4: Callback functions used by the ROS serial node on the Teensy 3.2.

The total time to execute each of the two callback functions was analysed by recording the execution time over 100 samples and calculating the mean and maximum times as shown in Table 6.2. Most of the execution time of the sensor data retrieval callback is waiting for the Teensy LCs to respond.

Table 6.2: Execution time of the callback functions performed by the Teensy 3.2.

Callback Function	**Mean** (μs)	**Maximum** (μs)
TargetAngle	1218	1221
SensorDataRetrieval	15467	15547

A test program not running the ROS serial node was created to allow for testing of the snake robot without relying on ROS. The program has the same functionality as the ROS version but generates its own motor angles to allow the testing of new gait patterns. The program communicates with the same version of the Teensy LC software for ease of use.

6.4 Snake Robot Module Software Design

The Teensy LC is the microcontroller contained in each module whose function is to retrieve sensor data and send it back to the Teensy 3.2 both reliably and as fast as possible. The software was written so that the same software can be used on all of the modules without any modifications. A modular design is used to allow for additional functionality to be added without regard to existing software.

The IMUs were calibrated as described in Chapter 3 using a least squares method. The calibration matrix and bias for each accelerometer takes the form as shown in Equation 6.1 [40]. As the calibration matrix is unique to each device, the values are stored in the EEPROM of the Teensy LCs to allow for the same software to be used for each device.

$$\boldsymbol{\alpha}_b = \mathbf{M}_{cal}(\boldsymbol{\alpha}_i - \mathbf{b}_{cal}) \tag{6.1}$$

The Teensy LC is unable to perform multitasking due to the ARM Cortex-M0+ processor it uses, so the software was designed to handle performing different tasks at the correct time. Due to time critical tasks, a simple polling system was unable to reliably perform tasks at the desired time.

A Real Time Operating System (RTOS) could be used to simulate multitasking which would fix the timing issues, though adds complexity and significant overheads to the system. The current system has a relatively small number of tasks that need to be performed such that a full RTOS is unnecessary. A task scheduler designed by Arkhipenko [64] for Arduino boards is utilised as the best solution for the project needs.

The task scheduler offers cooperative multitasking for Arduino microcontrollers where each task runs at a specified rate. Each task is completed before any other task can be run as there is no pre-emption between tasks, therefore each task needs to be designed to take as little time as possible so that time critical tasks have an opportunity to run. The tasks used are the IMU data retrieval, the Madgwick algorithm, save sensor data, send motor angles and sensor data request, which each runs a callback function shown in Figure 6.5 when called.

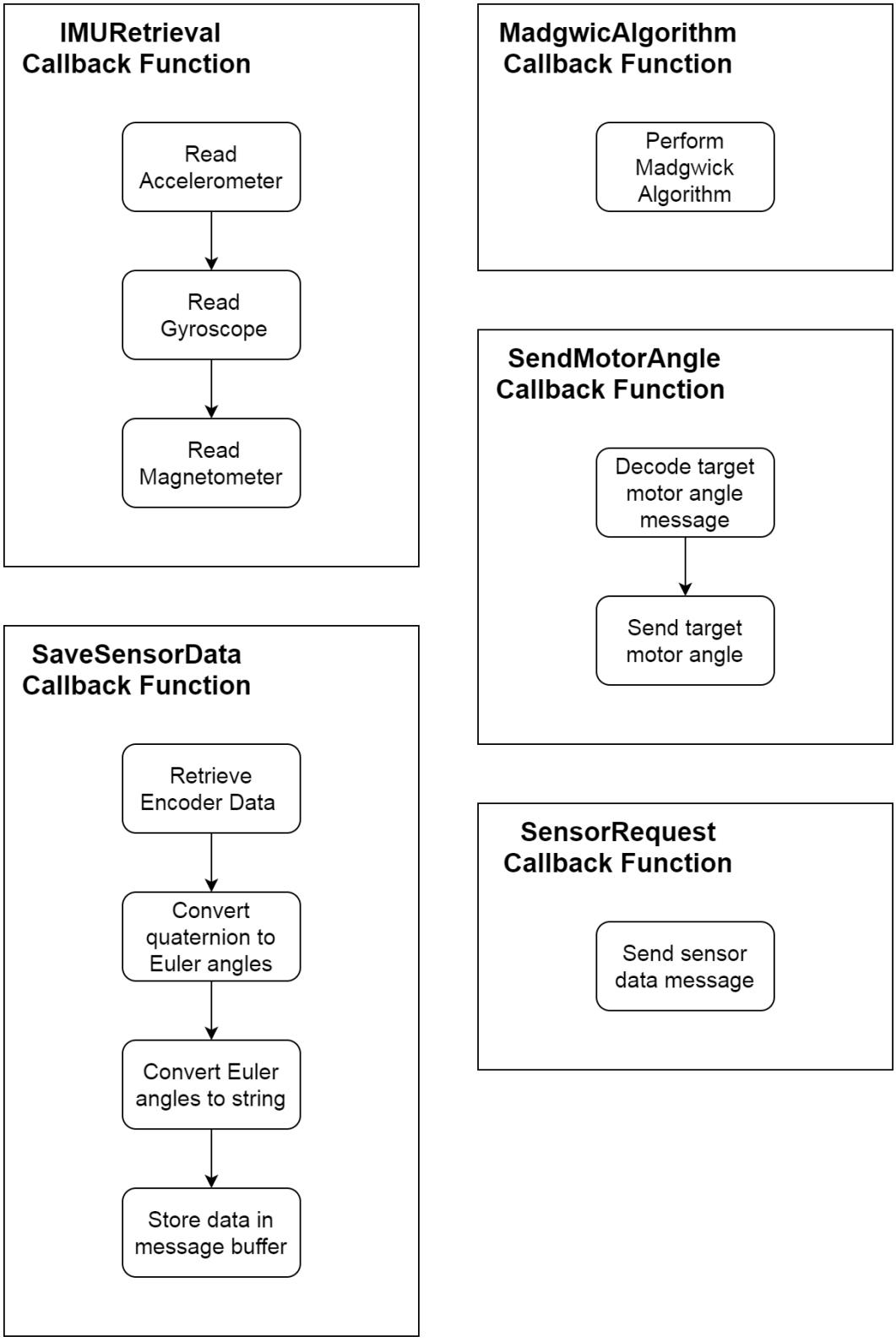


Figure 6.5: Callback functions used by the Teensy LC task scheduler.

Tasks can be set to have different priority levels such that higher priority tasks will always run over lower priority tasks if they are both waiting. Figure 6.6 shows the priority order of the tasks run by the Teensy LC. The task scheduler determines which task to run next in a specific order. The task scheduler first checks if the high priority tasks are ready to run then checks the normal priority tasks. If more than one task is ready in the same priority level, the task scheduler will run the task that has been waiting longest.

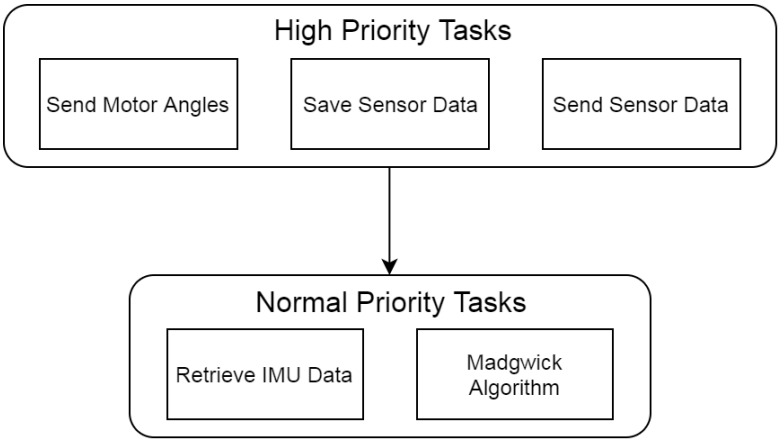


Figure 6.6: Priority of tasks used by the Teensy LC task scheduler.

The IMU retrieval task collects the accelerometer, gyroscope and magnetometer data from the MinIMU-9 v5. The accelerometer data is corrected using the calibration matrix stored in the EEPROM and scales the data according to the full-scale range used which in this case is 8 G. The gyroscope data is scaled according to its full-scale range set to 2000 DPS. The magnetometer data for the head module is calibrated using the calibration matrix and scaled using the full-scale range set to 4 gauss.

The Madgwick algorithm task calculates the quaternions representing the orientation of the module, using an analytically derived and optimised gradient-descent algorithm [39]. The filter can optionally use a magnetometer for gyroscope bias drift compensation. The advantages over the more common Kalman filter are being computationally inexpensive and is effective at low sample rates.

The IMU retrieval task and the Madgwick algorithm task are both normal priority tasks that run at a specified loop rate. The IMU data retrieval task runs at 100 Hz so therefore the Madgwick algorithm runs at 1000 Hz as it has to run at 5-10 times the retrieval rate to converge.

The three tasks involving communication over the RS485 data bus have a higher priority level due to being time critical. When a message is received by the Teensy LC, an interrupt is triggered which is immediately serviced. The RS485 data is sent and received using the same format as the Teensy 3.2 so that the Teensy LC can check the receiver address to determine if any action is required. If the message is for the device, the Teensy LC uses the purpose command to enable one of the three time critical tasks. The task that was interrupted is then completed followed immediately by the high priority task. Using this system, the time critical tasks are always run as soon as possible.

The send motor angles task sends the angle sent from the Teensy 3.2 to the servo motor. The task first decodes the message to extract the target angle for its device then sends the target angle to the servo motor with an execution time of 100 ms. If the motor is in a yaw module, the motor LED is set to green; it is set to blue for the pitch modules. The head module does not have a motor and as no motor angle corresponds to module zero, the task is effectively ignored.

The save sensor data task prepares the sensor data to be sent through the RS485 data bus to Teensy 3.2. When the save request is received from the Teensy 3.2, the normal priority tasks are disabled and a timer is started to ensure that the normal priority tasks are re-enabled if the data request is not received. The normal priority tasks are disabled so that when the data request is received, the message can be sent quickly without waiting for other tasks to finish. The disadvantage of this approach is that the more modules added, the longer the tail end modules have the normal priority tasks disabled for. Using the 10 modules on the University of Canterbury snake robot, no adverse effects are observed by the normal priority tasks being disabled.

The joint angle is retrieved from the motor encoder, then the quaternions from the most recent Madgwick update are converted to Euler angles and finally converted to a string for sending. The three Euler angles and the joint angle is stored in the message buffer ready to be sent.

The send sensor data task sends the sensor data prepared through the RS485 data bus to the Teensy 3.2. When the sensor request is received, a sending timer is started, to ensure that the device sends the data in the allotted time and does not try to send data while another device is sending data. After the message is sent, the normal priority tasks are enabled and the save timer disabled.

The task scheduler does not offer preemptive multitasking meaning that a task has to finish before the next one begins. The tasks should therefore be as short as possible to allow the high priority tasks to run more quickly after being enabled. The execution time of all the tasks was profiled by sampling 100 executions and finding the mean and maximum execution times. Table 6.3 shows the execution times.

Table 6.3: Execution time of the five tasks performed by the Teensy LC on module 3.

Task	**Mean** (μs)	**Maximum** (μs)
Retrieve IMU Data	1216	2290
Madgwick Algorithm	1019	1954
Send Motor Angles	1094	1097
Save Sensor Data	3612	4248
Send Sensor Data	790	792

The data save request timer duration in the Teensy 3.2 was set based on the maximum possible time to save the sensor data of modules zero. The worst case from Table 6.3 is if the save request is received at the start of the retrieve IMU data task, which would give a maximum save time of 6538 μs. Module zero is the head module and therefore does not have a motor so does not retrieve the encoder data which takes approximately 2000 μs. The save request timeout was chosen to be 6 ms ensuring that all the modules have time to save and prepare the sensor data. The sensor request timeout on the Teensy 3.2 was set to 1 ms based on the maximum execution time of the send sensor data task with additional time given to determine the purpose of the received RS485 message to start the task. The save timer was set to 16 ms based on the maximum execution time of the sensor data retrieval of the Teensy 3.2.

6.5 Pose Estimation Software

The pose estimation algorithm presented in Chapter 5 is performed by the snake UKF node, which subscribes to three topics: *"interface_stop"*, *"interface"* and *"sensor_feedback"*. Each topic has a callback function as seen in Figure 6.7, which is called when a message is received on the topics.

When the snake UKF node is first started, the UKF is initialised to zero for all the state vector parameters except the scalar part of the quaternion which is set to one.

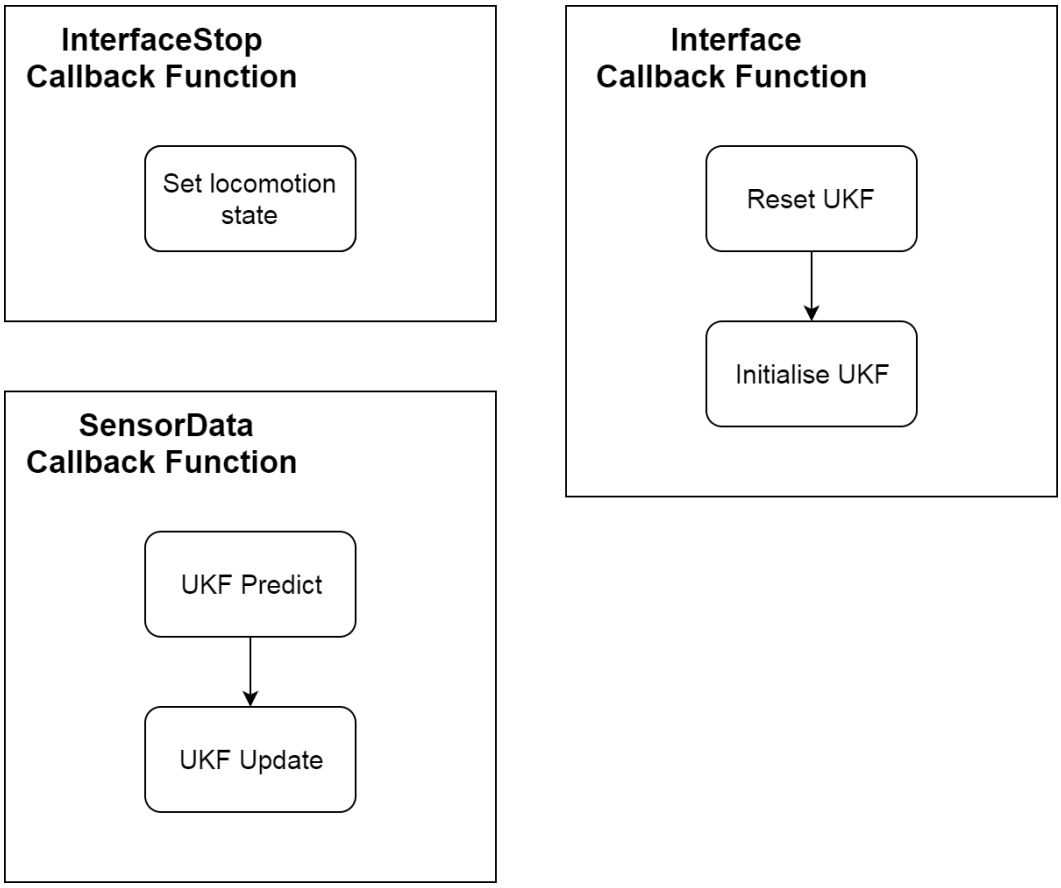


Figure 6.7: Callback functions used by the ROS Snake UKF node.

The state covariance is set to one and the process and measurement covariance are set to their predetermined values.

The *SensorData* callback function performs the UKF algorithm based on the Kalman filter library written by Herb [65]. The library was modified to use the SR-SSUKF with quaternions in the state vector as outlined in Chapter 5. The UKF predict step is performed first followed by the update step using the received sensor data.

The predict step computes the sigma points and runs each set through the process model to get the predicted state and sigma state points. The average of the sigma state points is found based on the sigma weights and used to predict the state covariance.

The update step first detects if there are any failed sensor readings and if so, sets the measurement covariance for that parameter to the sufficiently large value of 10^6 such that the sensor reading is effectively ignored. The measurement vector

and corresponding sigma measurement points are predicted using the sigma state points. The square root innovation covariance is found which is then used to find the Kalman gain. The state is updated and after that the state covariance calculated.

The execution time of the UKF was determined by sampling 100 executions and finding the mean and maximum execution times which are shown in Table 6.4. This shows that the algorithm is efficient enough to be performed every gait cycle.

Table 6.4: Execution time of the Sensor Data callback function in the snake UKF node.

Callback Function	**Mean** (μs)	**Maximum** (μs)
SensorData	1012	1081

The *InterfaceStop* callback function sets a flag to change the state to moving or not moving. The flag is used in the *Interface* callback function which prevents the state vector from being changed if the robot is moving. If the snake robot is not moving, the state vector is reset and initialised with the new gait parameters received.

A data logger was used to output the predicted state, predicted measurements, the sensor data and the updated state to a text file which was used for testing.

6.6 Summary

The University of Canterbury snake robot uses ROS to perform the UKF for pose estimation using sensor data collected from the Teensy LCs on each modules through the Teensy 3.2. ROS uses five nodes, gait generation node, GUI node, serial node, snake timing node and snake UKF node.

The GUI and gait generation nodes were written by Gilani and perform the same functions as the previous prototype. The GUI allows the user to select the gait pattern or input raw motor angles, which the gait generation node uses to generate motor angles to be sent to the snake robot. The message structure of the target motor angles message was updated to use an array instead of individual variables.

The snake timing node sends a sensor data request to the serial node at a rate of 10 Hz to start the pose estimation loop. The serial node was rewritten to send the motor angles through the RS485 data bus and retrieve the sensor data from the

Teensy LCs. The Teensy 3.2 running the serial node is the master device on the RS485 data bus. The structure of the RS485 messages contains the sender device address, the receiver device address, the message purpose command and a data buffer where necessary.

The motor data is sent in a single message when the motor target angle callback function is called to ensure that the motor data is received by the Teensy LCs at the same time. When the sensor data retrieval is called, a save request to all the devices is sent followed by a 6 ms timeout to allow the first modules time to save the sensor data. The sensor data from each device is requested sequentially and stored in the sensor feedback message buffer until all the data is retrieved and the message is sent.

The Teensy LC uses a task scheduler with multiple priority levels to allow for time critical tasks to be executed. The two normal priority tasks are the IMU retrieval and the Madgwick algorithm which run at 100 Hz and 1000 Hz respectively. The three high priority tasks are send motor angle, save sensor data and the sensor data request. When a message is received on the RS485 data bus an interrupt is triggered which reads the receiver address and purpose command and starts the commanded task if it is the receiver device.

The snake UKF node performs the pose estimation algorithm presented in Chapter 5. The UKF predict step is first performed followed by the update step using the received sensor data.

Chapter 7

Results and Verification

This chapter presents the verification of the pose estimation algorithm using the Vicon motion tracking system to provide a ground truth to compare to the SR-SSUKF prediction. Each of the gaits outlined in Chapter 5 were tested to determine the accuracy of the modules orientation prediction. Additional tests were conducted to test the algorithm starting in a different orientation and to test module failure accuracy.

7.1 Verification Setup

To test the accuracy of the pose estimation algorithm, a Vicon motion tracking system was used. The Vicon system uses four retroreflective markers captured by four cameras to track a rigid body and determine its orientation and position. The Vicon system can simultaneously track multiple rigid bodies as long as the markers for each body are far enough apart that they do not interfere with each other. The snake body is comprised of 10 connected rigid bodies, but are too close together, making tracking all the modules simultaneously difficult. To simplify the setup of the Vicon system, only three modules were tracked: the head module (module 0), the tail module (module 9) and module 4. The retroreflective markers used to track each of these modules can be seen in Figure 7.1. Due to the size of the trackers, there was not enough space to attach the markers in such a way that the rolling gait could be performed, so was omitted from the verification tests.

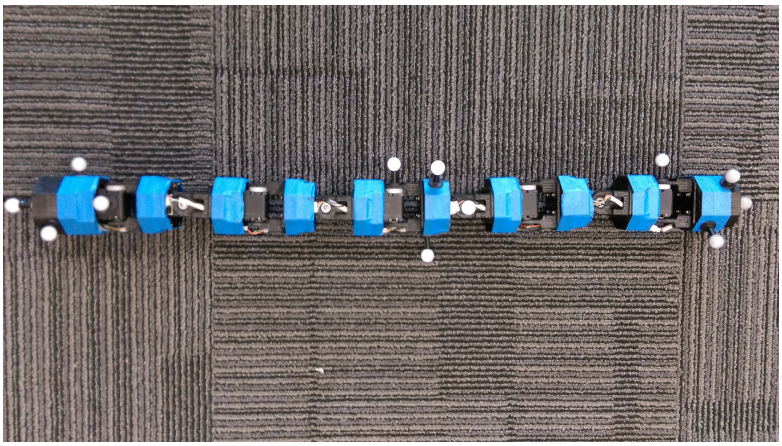


Figure 7.1: Setup of the retroreflective markers used by the Vicon motion tracking system for the three modules.

The Vicon motion tracking software used to output the orientation and position of each rigid body can be seen in Figure 7.2. The coordinate system used by the Vicon system was different than the pose estimation algorithm so quaternions were outputted and converted to the coordinate system of the pose estimation algorithm, allow proper comparison.

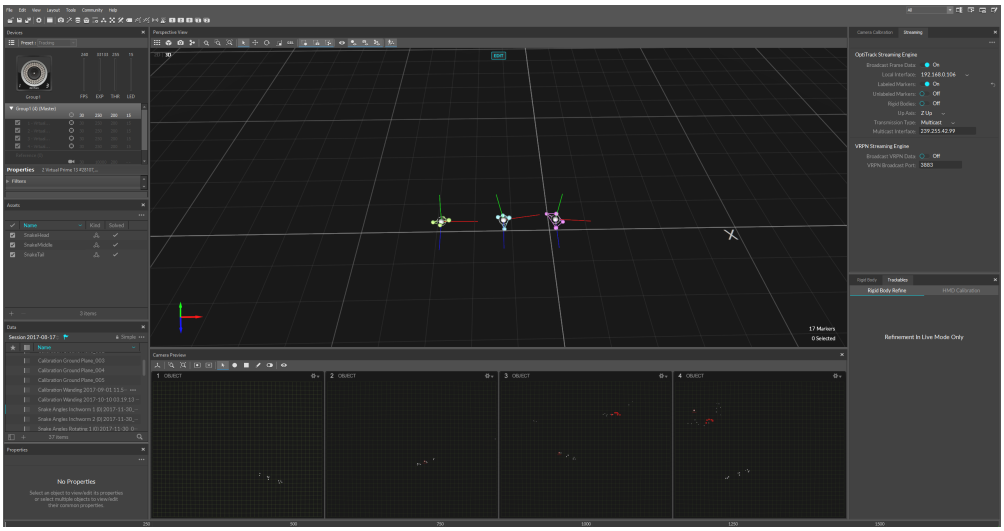


Figure 7.2: The Vicon motion tracking software.

The Vicon system was setup in a different building to where the internal testing was conducted so the IMUs were recalibrated in the Vicon room. The Vicon system outputs the motion tracking data at a faster rate than the pose estimation algorithm such that the two data sets were not synchronised. To synchronise the two sets of data, the Vicon data was interpolated into the same data rate as the pose estimation algorithm and a cross correlation between the Vicon pitch and the sensor pitch of the head module used. The sensor data for the pitch was used

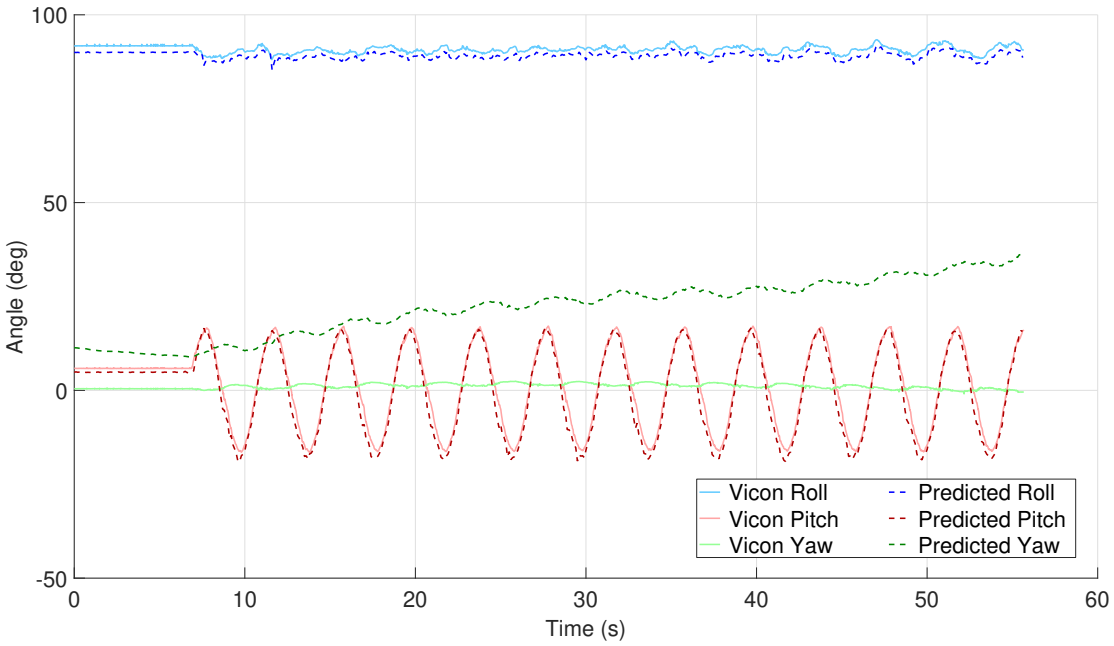
instead of the predicted pitch due to the small delay between the sensor data and the prediction shown in the internal tests in Chapter 5.

7.2 Verification Results

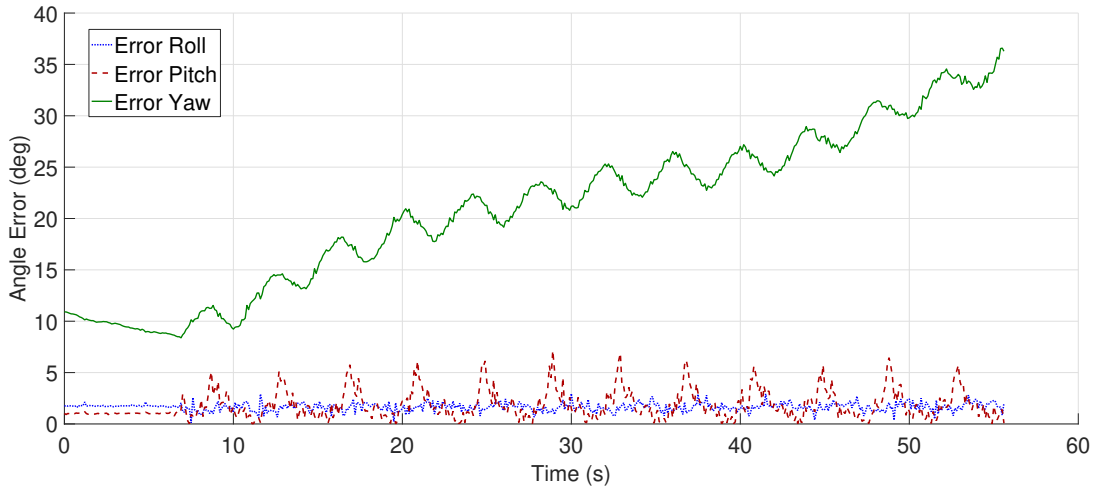
Each of the tests was conducted by initialising the selected gait with the head module set to approximately 0 degrees yaw in the Vicon system. The Vicon motion tracking started recording after the snake robot was initialised and the snake motion was started.

7.2.1 Motion Gait Tests

Each of the different motion gaits were tested using the Vicon motion tracking system using their respective gait parameters model. Figure 7.3a shows a comparison between the predicted Euler angles for the head module using the SR-SSUKF pose estimation algorithm compared to the Vicon system while the snake robot is using the linear progression gait. It can be seen that the predicted roll and pitch closely follow the Vicon Roll and Pitch but the predicted yaw is steadily diverging from the Vicon yaw which can clearly be seen in the prediction error in Figure 7.3b.



(a) The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the linear progression gait.



(b) The prediction error of the head module using the linear progression gait.

Figure 7.3: The pose estimation algorithm compared to the Vicon motion tracking system for the head module while performing the linear progression gait.

The yaw prediction is based on a single magnetometer in the head module of the snake robot which is used to correct for gyroscope drift in the IMU. The yaw sensor data from the head module was likely being effected by metal under the floor and even though the magnetometer was calibrated in the Vicon room, there was too much magnetic interference for the magnetometer to correct the gyroscope drift. Figure 7.4 shows the yaw prediction of the head module for the linear progression gait including the first 30 seconds before the motion gait was started. It can be seen that while stationary on the point that it was calibrated on, the yaw prediction stays relatively stable. Only after the motion gait starts does the prediction trend upwards. This supports the theory that magnetic interfere is effecting the yaw prediction.

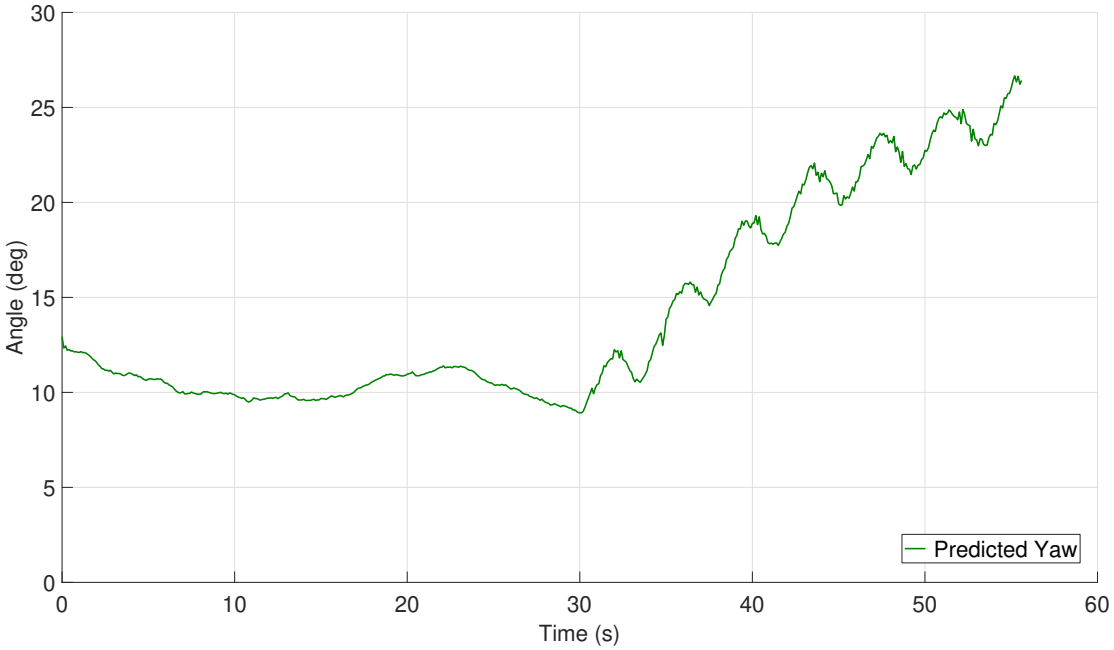


Figure 7.4: The predicted yaw of the head module using the linear progression gait.

The turning gait shown in Figure 7.5 uses the same model as the linear progression but includes turning and as such has similar prediction accuracy. The rotating gait shown in Figure 7.6 was the most complex gait tested. The motion of the modules are more variable than in the internal tests due to the weight of the markers unbalancing the modules. However the predicted roll and pitch can be seen to be accurately following the Vicon roll and pitch despite the variable motion. The yaw prediction in the rotating gait has the same oscillatory motion as the Vicon yaw though offset by approximately 70 degrees after the initial error in the trend at the start of the snake robot motion.

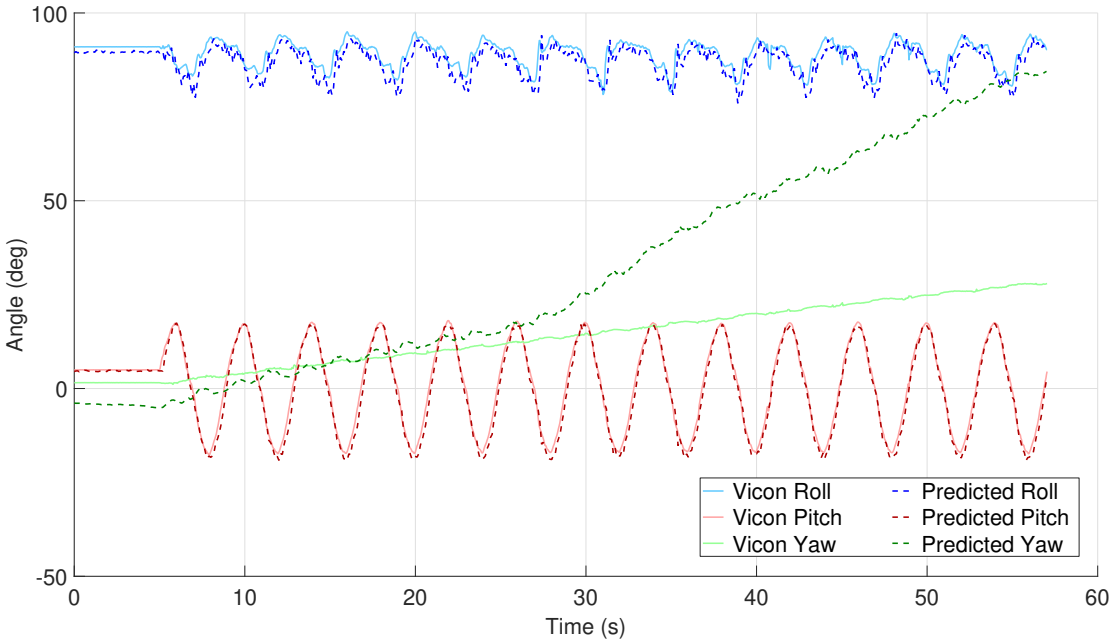


Figure 7.5: The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the turning gait.

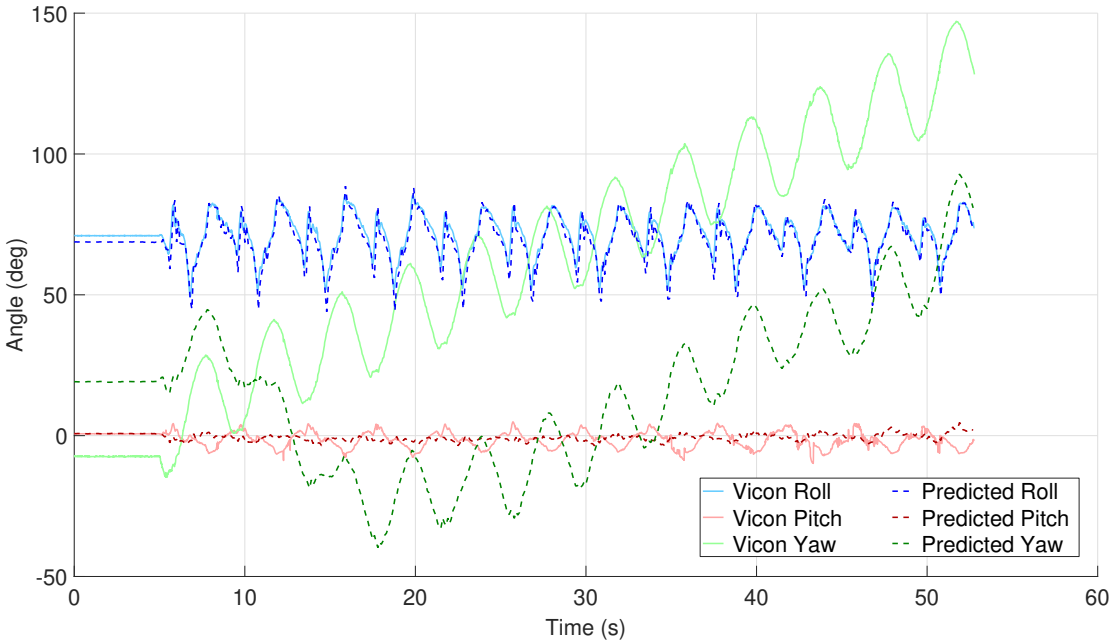


Figure 7.6: The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the rotating gait.

The joint angles model, which uses a third order model to directly predicted the joint angles, was tested using the the linear progression and rotating gaits to compare the accuracy of the different model types. The comparison between the pose estimation algorithm and the Vicon system for the two gaits can be seen in Figure 7.7 and Figure 7.8. It can be seen that the linear progression roll prediction is not as accurate as the equivalent gait parameter model but the rotating gait has similar prediction accuracy.

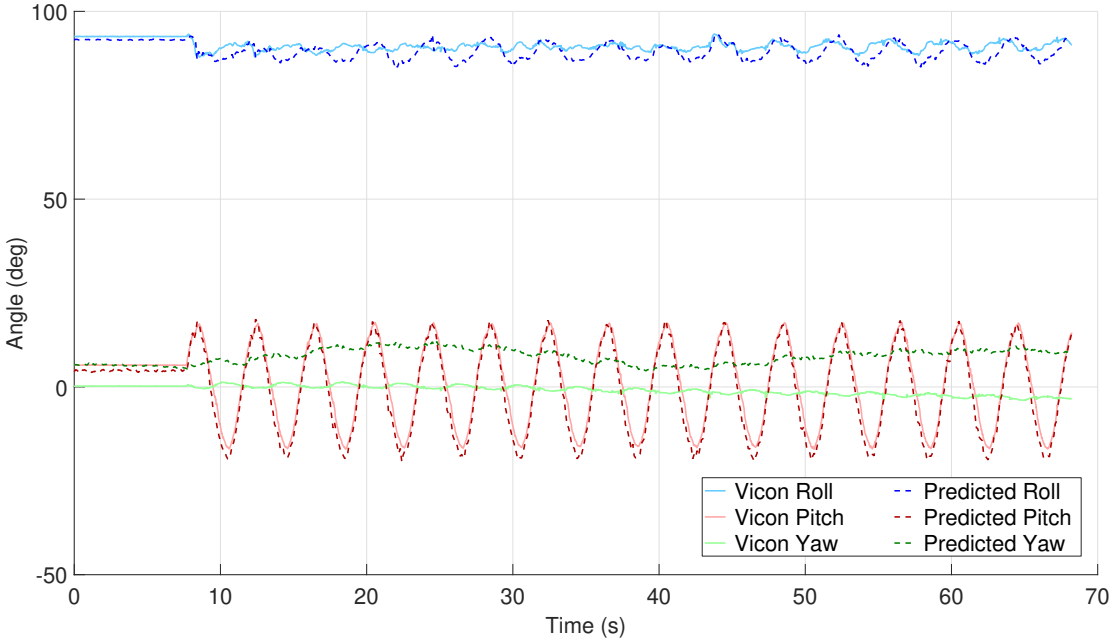


Figure 7.7: The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the linear progression gait with the joint angles model.

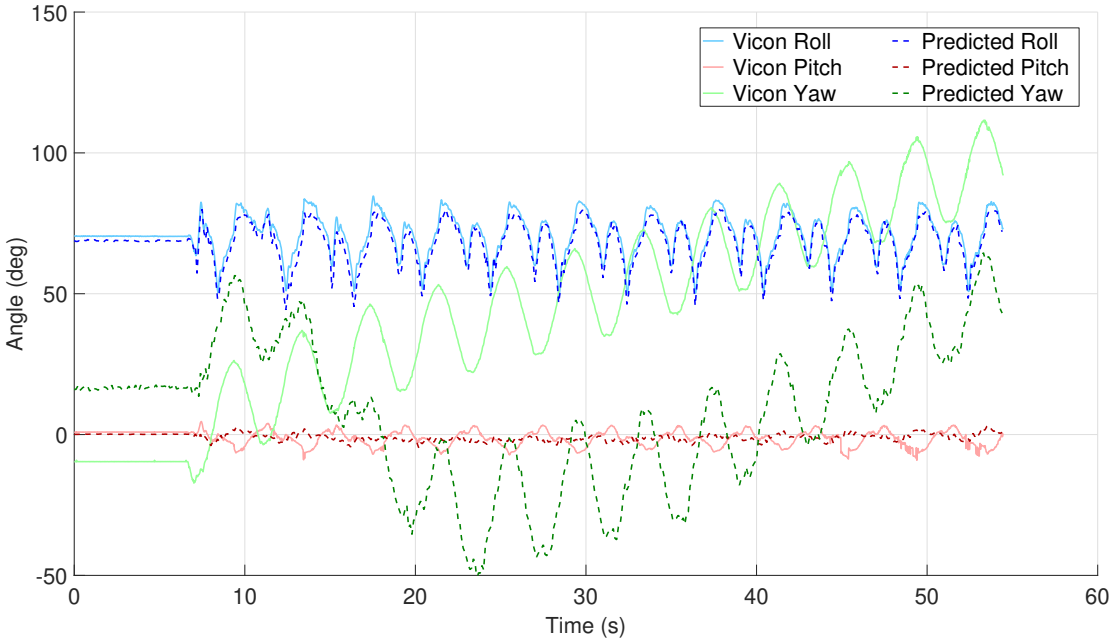


Figure 7.8: The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the rotating gait with the joint angles model.

The mean Euler angles error from each of the tests for the head module were calculated and can be seen in Table 7.1. The linear progression gait is more accurate than the rotating gait as expected since the rotating gait is more complex. It can be seen that the gait parameter models and the joint angles model used for pose estimation perform similarly with the gait parameters model being slightly more accurate. The joint angle model however can perform all of the motion gaits tested with the same process model whereas the gait parameter models use a separate model for each motion gait. With similar accuracy between the two models, the joint angle model is more practical to use since it is not reliant on knowledge of the motion gait being used and allows for development of additional gait patterns without requiring new process models to be developed for the pose estimation algorithm.

Table 7.1: Vicon mean Euler angles error for the head module.

Gait	**Roll Error (Degrees)**	**Pitch Error (Degrees)**	**Yaw Error (Degrees)**
Linear Progression	1.65	1.82	20.92
Turning	2.29	1.36	19.12
Rotating	2.34	2.59	54.80
Angles Linear Progression	1.84	2.12	8.93
Angles Rotating	2.58	2.51	49.34

7.2.2 Further Tests

The linear progression gait was tested moving down a ramp angled at 15 degrees to test the pose estimation algorithm in a different orientation. Figure 7.9 shows the Euler angles comparison for the head module as it moves down the ramp using the gait parameters model. The pitch is centered around 15 degrees and then trends towards zero as the snake robot moves off the ramp onto flat ground. Table 7.2 shows the mean Euler angles error of the head and tail modules where it can be seen that the pose estimation algorithm is only slightly less accurate than on flat ground, such the the algorithm is still able to predict the roll and pitch of the modules at a different orientation with similar accuracy.

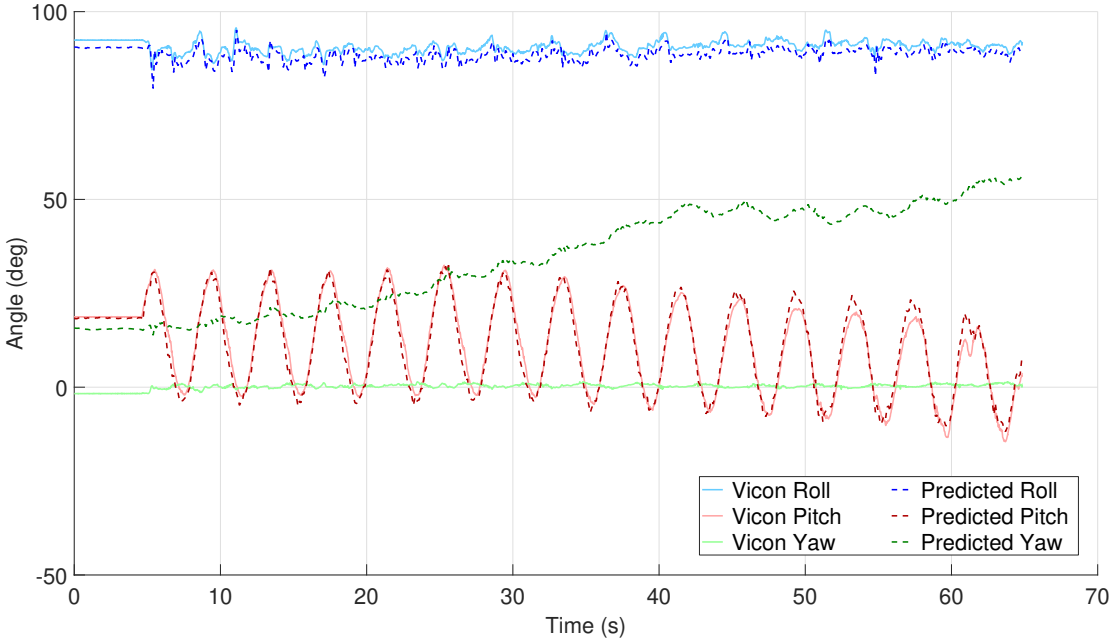


Figure 7.9: The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the linear progression gait while moving down a 15 degrees ramp.

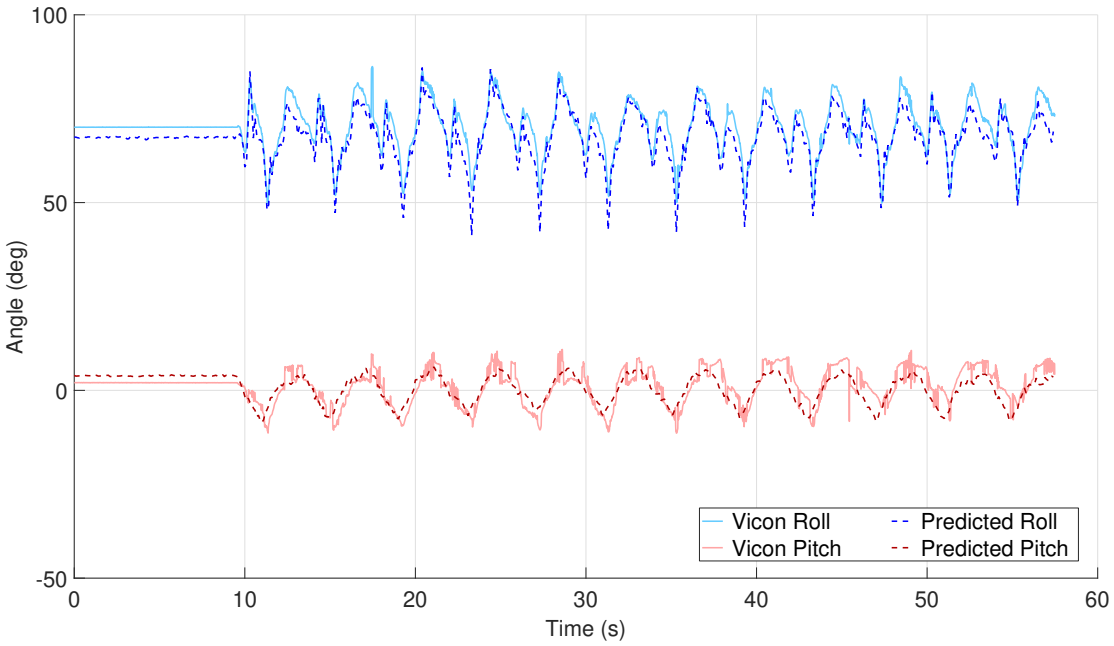
Table 7.2: Vicon mean Euler angles error for the head module performing the linear progression gait while moving down a ramp.

Module	**Roll Error (Degrees)**	**Pitch Error (Degrees)**	**Yaw Error (Degrees)**
Head	2.06	1.79	33.94
Tail	1.71	2.17	34.18

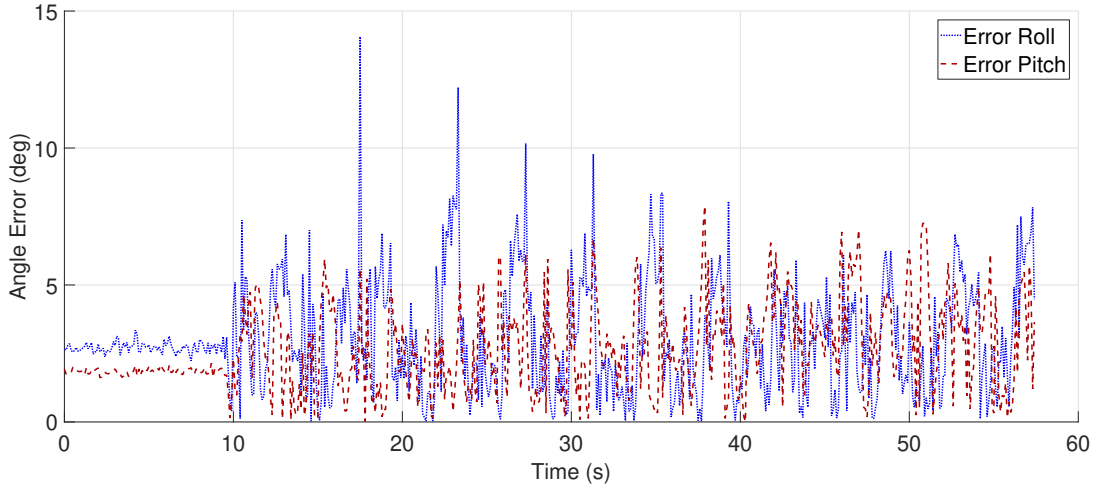
7.2.3 Module Failure

In ordinary operation, sensor data occasionally gets dropped due to communication errors so no data is received by the base station. The base station sets the sensor data to 200 to represent a sensor retrieval failure. To test the effectiveness of the sensor error handling, the sensor data from module 4 was disabled so that the base station does not receive any sensor data from module 4. The rotating gait was performed using the gait parameters model to test the accuracy of the pose estimation with a sensor failure. As the predicted yaw has been shown to be inaccurate in previous tests, the yaw angles are not shown for the module failure tests to improve the clarity of the data. Figure 7.10 shows the prediction comparison for module 4 with both the Eulers angles and the prediction error. It can be seen that despite having no sensor data from module 4, the pose estimation is still able to accurately predict the orientation of the module.

The pose estimation algorithm was tested for the case where one of the motors stops working. To simulate a hardware failure, the motor in module 4 was set to a constant angle of 0 degrees while the rotating gait is performed. Figure 7.11 shows the Euler angles comparison and prediction error for module 4 in which it can be seen that the roll is predicted accurately but with occasionally prediction spikes. The pitch is fairly accurate in absolute error but the trend is not accurately following the actual pitch from the Vicon system. To see the effects of the failed motor on the rest of the snake robot, Figure 7.12 shows the comparison for the head module. The head module has similar prediction accuracy to module 4 with corresponding prediction spikes in the roll and the pitch not accurately predicting the oscillatory motion seen in the Vicon pitch.

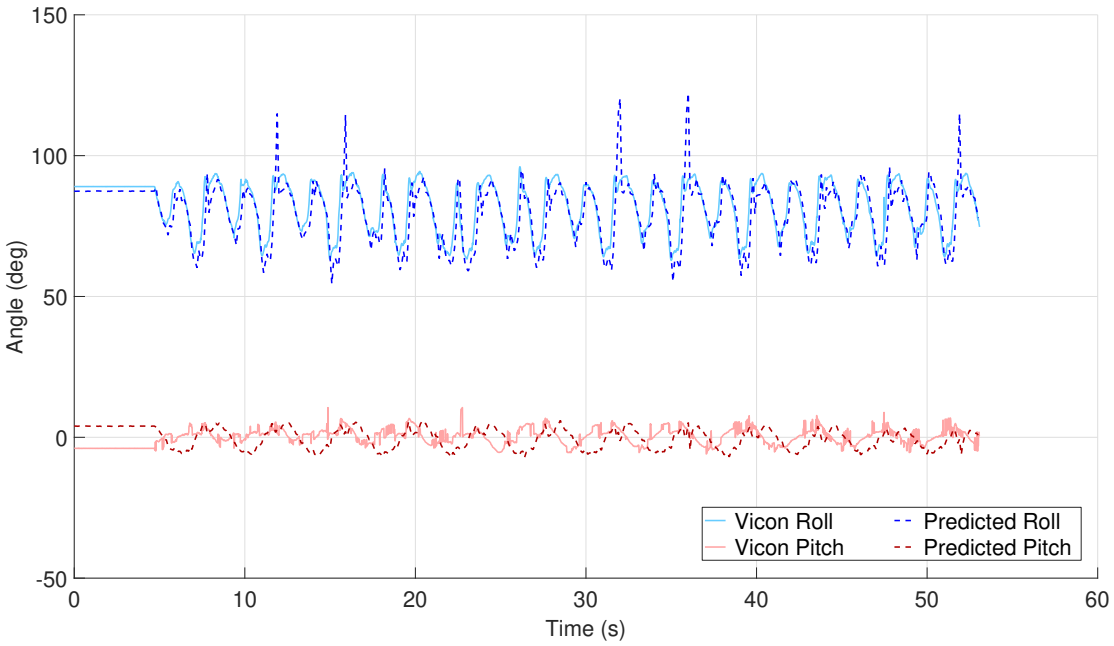


(a) The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the module 4 using the rotating gait.

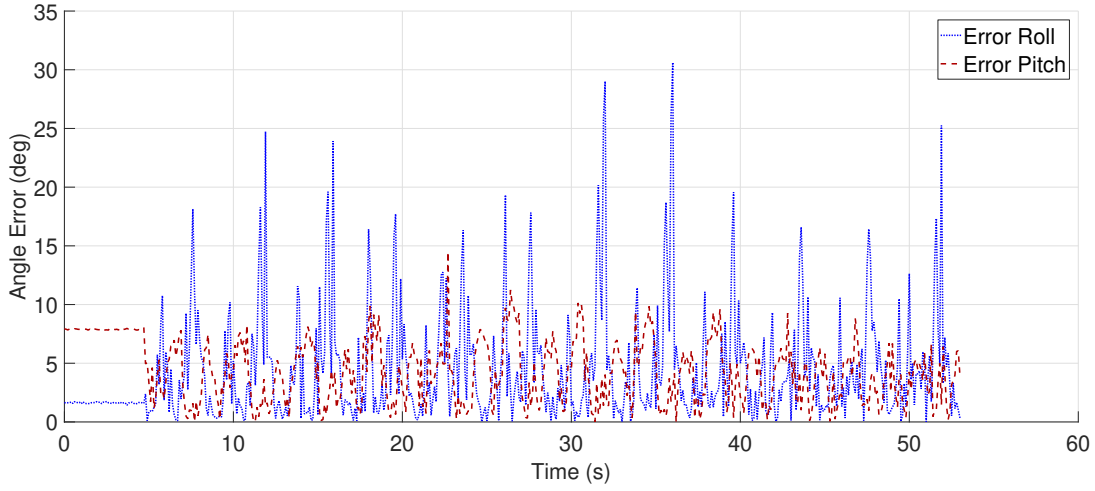


(b) The prediction error of module 4 using the rotating gait.

Figure 7.10: The pose estimation algorithm compared to the Vicon motion tracking system for the module 4 while performing the rotating gait where the sensor in module 4 has failed.

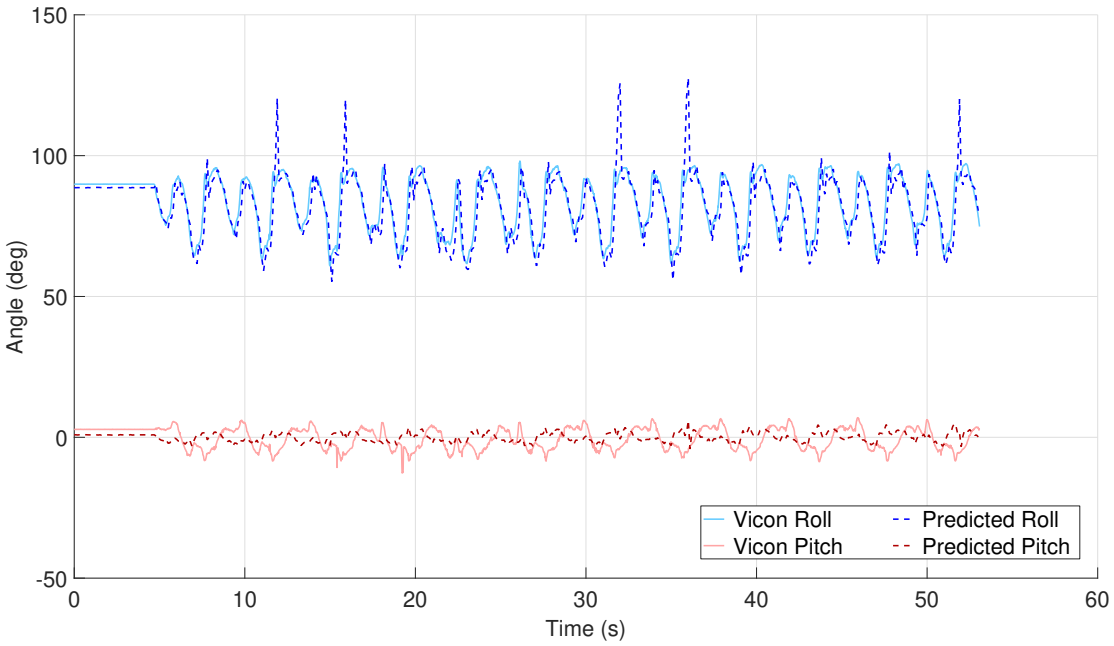


(a) The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the module 4 using the rotating gait.

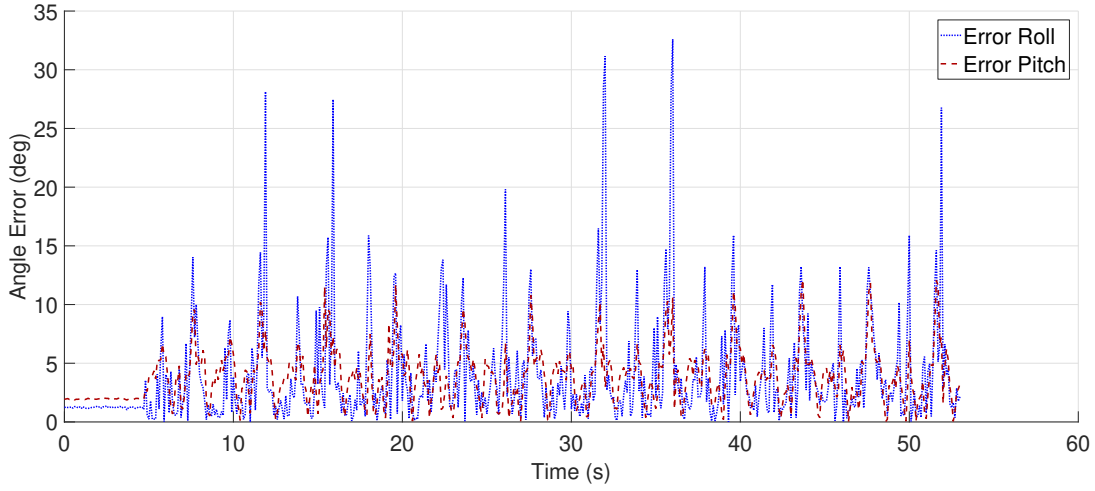


(b) The prediction error of module 4 using the rotating gait.

Figure 7.11: The pose estimation algorithm compared to the Vicon motion tracking system for the module 4 while performing the rotating gait where the motor in module 4 has failed.



(a) The Euler angles comparison between the predicted pose and the Vicon motion tracking system for the head module using the rotating gait.



(b) The prediction error of the head module using the rotating gait.

Figure 7.12: The pose estimation algorithm compared to the Vicon motion tracking system for the head module while performing the rotating gait where the motor in module 4 has failed.

Table 7.3 shows the mean Euler angles errors for the module failure tests. It can be seen that the sensor failure of one module does not adversely effect the accuracy of the pose estimation algorithm, however the failure of the motor in module 4 did have a small effect on the accuracy of the pose estimation algorithm.

Table 7.3: Vicon mean Euler angles errors for the module failure tests.

Failure	**Module**	**Roll Error (Degrees)**	**Pitch Error (Degrees)**
No Failure	0	2.34	2.59
No Failure	4	2.89	2.04
No Failure	9	4.02	1.97
Sensor Failure	0	2.29	2.26
Sensor Failure	4	3.02	2.62
Sensor Failure	9	3.36	1.68
Hardware Failure	0	3.89	3.81
Hardware Failure	4	4.42	4.33
Hardware Failure	9	4.59	3.68

7.3 Summary

The pose estimation algorithm present in Chapter 5 was verified using a Vicon motion tracking system to give a ground truth of the orientation of the head module, the tail module and module 4. Each of the motion gait described in Chapter 5, except the rolling gait, was tested to determine the accuracy of the predicted Euler angles of the modules. From Table 7.1 it can be seen that the roll and pitch of the head module are predicted well for all of the motion gaits.

The yaw prediction however is poor and inconsistent as demonstrated by the range in the mean error from 8.93 degrees from the joint angles model using the rotating gait to the 54.80 degrees from the gait parameter model using the rotating gait. The yaw prediction is poor due to being based on the magnetometer in the head model which is used to correct for the gyroscope drift and generate yaw sensor data. Magnetometers are susceptible to changes in the magnetic field due to the presence of metal and though it works relatively well in the development room, the Vicon room had too much magnetic interference to work reliably.

The joint angle model and the gait parameter models were compared. With the setup of the test, both models accuracy was very similar with the gait parameter model only being slightly better. The joint angle model can perform all of the

motion gaits tested with the same process model since it is not reliant on prior knowledge of which gait pattern is to be used, so it is the more practical choice of model as further gait patterns are developed.

The linear progression motion gait was tested moving down a ramp at 15 degrees to test the pose estimation algorithm with the snake robot in a different orientation. From Table 7.2 it can be seen that the average roll and pitch prediction error is only slightly larger than on flat ground and shows that the pose estimation algorithm can operate at different orientations.

One of the features of a snake robot is the redundant design that allows continued operation even if a module fails. Two tests were conducted to test the redundancy of the pose estimation algorithm: a full sensor failure of a module and a hardware failure of a motor. The module failure tests were conducted on module 4 with first the sensor data being set to not return while the robot was performing the rotating gait. Table 7.3 shows that the prediction data for module 4 is still accurate even without the sensor data. The Herkulex servo motor was then set to a constant 0 degrees to simulate a hardware failure. From the results in Table 7.3, the average prediction error for the hardware failure was slightly higher than without a module failure.

The verification of the pose estimation algorithm using the SR-SSUKF, has been shown to work accurately for the roll and pitch of each module across each of tested gaits, and works well in the case of sensor failure but is slightly effected by hardware failures. The yaw prediction being tied to the magnetometer in the head module is poor and a better solution should be sought to improve the yaw sensor data.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

A pose estimation algorithm was developed for feedback control in a snake robot previously designed at the University of Canterbury. Each module of the snake robot contained a Herkulex DRS-0101 smart servo which has an inbuilt motor encoder. Additional components were added to collect sensor data and send it to the PC: A MinIMU9-v5, Teensy LC, and MAX3485. The MinIMU9-v5 was the low cost IMU chosen which was calibrated using a least squares method after being mounted in each module. The snake robot is tethered so the RS485 hardware standard was selected as the data bus to transfer the sensor data from each of the modules to the base station connected to the PC. At the expected tether length of 100 m, the RS485 standard has a speed of up to 7 Mbps. The MinIMU9-v5 does not support RS485 so a Teensy LC with an attached MAX3485 integrated circuit was added to each module to retrieve the sensor data from the IMU and encoder and send it to a Teensy LC chosen for the base station.

The pose estimation algorithm was based off previous ROS software written for the snake robot that contained the gait generation node and the GUI node. The gait generation node takes the parameters selected in the GUI and generates the motor angles at each time step using the sinusoid function generator. Three nodes were developed for the pose estimation algorithm: the snake UKF node, the snake timing node and the serial node. The snake UKF node runs the SR-SSUKF after

sensor data has been received, and the snake timing node sends a sensor request at a rate of 10 Hz.

The sensor data is collected using the serial node running on the base station. The base station is the master device on the RS485 data bus and sends the motor angles and retrieves the sensor data from each module. The structure of the RS485 message is designed to contain the sender address, receiver address, message purpose and a data buffer. The sensor data is collected in two stages, first a save request is sent to all the modules to save the current sensor data and then a request for that data is made in sequence. This approach ensures that all of the sensor data collected has the same timestamp.

The Teensy LC runs a task scheduler with multiple priority levels to ensure that the time critical tasks of the sensor data collection are completed as quickly as possible. The software is designed to be modular so that the same program can be used on all the devices. The device ID and calibration data is stored in the EEPROM of each device.

The pose estimation algorithm uses a SR-SSUKF, which is a non-linear extension to the Kalman filter. A virtual chassis was used to abstract away from the internal shape of the robot to allow it to be treated as though it were a wheeled robot. The virtual chassis has a body frame whose origin lies at the centre of mass of the snake robot with the axes aligned along the principle moments of inertia. The virtual chassis is constructed using a body frame fixed to the head module which is placed at the origin. A standard kinematics model is used to calculate the pose of all the modules with respect to the head module body frame. A transformation matrix is then found to convert the pose of each module into the virtual chassis frame.

The linear progression, turning, rolling and rotating gaits were all used to test the pose estimation algorithm with a gait based model used for each. An additional joint angle model, which does not require prior knowledge of the gait being performed, was also created to compare the two different approaches. The process model for each of the gait parameter models uses a second order model for each of the parameters and the joint angle error was damped by a scalar of 0.95. The process model for the joint angles model uses a third order model for the joint angles. The quaternion in all of the process models uses a discrete-time update developed by Van Der Merwe and the angle velocity was assumed to have constant velocity. The Measurement model was similar for all of the models with the only difference

being how the joint angles were calculated. The filter was tuned experimentally to optimise the accuracy of the filter prediction.

The accuracy of the pose estimation algorithm was verified using a Vicon motion tracking system. The head module, tail module and module 4 were each tracked by the Vicon system with the orientation outputted as a quaternion. The Vicon used a different coordinate frame than the snake robot so the quaternion was converted into the same coordinate frame as the prediction data. Each of the gaits, except for the rolling gait, were tested to determine the accuracy of the pose estimation algorithm. The linear progression gait using the gait parameters had the lowest mean Euler angles errors with a mean roll error of 1.65 degrees and a mean pitch error of 1.82 degrees. The linear progression gait had the lowest mean error as expected as it was the simplest motion gait tested. The rotating gait using the gait parameters model had a mean roll error of 2.34 degrees and an mean pitch error of 2.59 degrees. The rotating gait is a more complicated motion gait than the linear progression motion gait and as such had a higher mean error. The yaw prediction for all of the models was inaccurate due to being based on the magnetometer mounted in the head module which was too adversely affected by magnetic interference from the building.

The gait parameter models and the joint angle model were compared and were found to have similar prediction accuracy. The joint angle model can perform all of the motion gaits tested with the same process model since it is not reliant on prior knowledge of which gait pattern is being used. It is therefore the more practical choice of model as further gait patterns are developed.

One of the features of a snake robot is a redundant design to allow for continued operation even if a module fails. Two tests were conducted to test the redundancy of the pose estimation algorithm: a full sensor failure of a module and a hardware failure of a motor. The sensor failure test was conducted on module 4 with the sensor data being set to not return while performing the rotating gait. The prediction data for module 4 was still accurate without the sensor data with a mean roll error of 3.02 degrees and a mean pitch error of 2.62 degrees. To simulate a hardware failure, the motor in module 4 was set to a constant 0 degrees while using the rotating gait. The mean prediction error of module 4 was slightly higher with a mean roll error of 4.42 degrees and a mean pitch error of 4.33 degrees. The head and tail modules were similarly affected by the hardware failure with slightly increased mean errors.

8.2 Future Work

8.2.1 Hardware and Software Design

The hardware design needs further development to improve the reliability and accuracy of the sensor data from the snake robot. To reduce the sensor data return time, the Teensy LC in each module could be replaced with a Teensy 3.2 which would decrease the processing time and allow for the baud rate of the data bus to be increased. The base station could be included in the tail module of the snake robot and communicate directly with the computer through RS485. This would allow for a separate short range data bus to be used for internal communication. A PCB incorporating all of the electronic components onto a single board would reduce the number of wires in each module and therefore improve the electrical reliability.

The current power distribution system for the Herkulex DRS-0101 smart servos causes a delay between the actuation of the motors at the tail of the robot and the motors at the end of the robot such that the motors are not in sync with each other. During testing of the pose estimation algorithm, a number of motors stopped working as they did not have enough torque to properly perform the motion gaits. Improving the power line with higher rated wire and up upgrading the servo motors to the the Herkulex DRS-0201 smart servo (which has more torque) would help fix the motor issues.

8.2.2 Pose Estimation Algorithm Design

The main improvement that can be made to the pose estimation algorithm is improving the yaw prediction of each module. The yaw prediction is entirely dependent on the magnetometer located in the head module, but is too adversely affected by magnetic interference from the environment to be reliable. A better approach to obtaining sensor data for the head module is needed for a more accurate prediction to be made. Adding additional sensors able to determine the yaw would greatly improve the functionality of the pose estimation algorithm.

The SR-SSUKF currently runs at the same rate as the motor commands of 10 Hz which limits the capabilities of the pose estimation algorithm. By increasing

the sensor collection rate and the pose estimation rate, the algorithm can more quickly respond to changes in the snake robots pose, improving the accuracy of the pose estimation algorithm.

The joint angles model was shown to be only marginally less accurate than the gait parameters model but can perform all the gaits with the same model, therefore future research should by conducted using this model. The current algorithm design assumes that only a single gait pattern will be used which is impractical for real-world applications. Future work could be undertaken to ensure that the joint angles model works correctly while changing gait patterns during operation.

Bibliography

[1] C. Gilani, "Prediciting the motion trajectories of a modular snake robot performing various gaits," Master's thesis, University of Canterbury.

[2] J. K. Hopkins, B. W. Spranklin, and S. K. Gupta, "A survey of snake-inspired robot designs," *Bioinspiration & biomimetics*, vol. 4, no. 2, p. 021001, 2009.

[3] J. GRAY, "The mechanism of locomotion in snakes," *Journal of Experimental Biology*, vol. 23, no. 2, pp. 101–120, 1946. [Online]. Available: http://jeb.biologists.org/jexbio/23/2/101.full.pdf

[4] "Snake locomotion," https://kids.britannica.com/students/assembly/view/171904.

[5] M. Walton, B. C. Jayne, and A. F. Bennett, "The energetic cost of limbless locomotion," *Science*, vol. 249, no. 4968, pp. 524–527, 1990.

[6] A. A. Transeth, K. Y. Pettersen, and P. Liljebäck, "A survey on snake robot modeling and locomotion," *Robotica*, vol. 27, no. 07, pp. 999–1015, 2009.

[7] W. Mosauer, "A note on the sidewinding locomotion of snakes," *The American Naturalist*, vol. 64, no. 691, pp. 179–183, 1930. [Online]. Available: http://www.jstor.org.ezproxy.canterbury.ac.nz/stable/2456894

[8] K. J. Dowling, "Limbless locomotion: Learning to crawl with a snake robot," Thesis, 1996.

[9] M. Mori and S. Hirose, "Three-dimensional serpentine motion and lateral rolling by active cord mechanism acm-r3," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1, Conference Proceedings, pp. 829–834 vol.1. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1041493

[10] ——, "Development of active cord mechanism acm-r3 with agile 3d mobility," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, vol. 3, Conference Proceedings, pp. 1552–1557 vol.3.

[11] H. Komura, H. Yamada, and S. Hirose, "Development of snake-like robot acm-r8 with large and mono-tread wheel," *Advanced Robotics*, vol. 29, no. 17, pp. 1081–1094, 2015. [Online]. Available: http://dx.doi.org/10.1080/01691864.2014.971054

[12] K. Ito and H. Maruyama, "Semi-autonomous serially connected multi-crawler robot for search and rescue," pp. 1–15, 2016. [Online]. Available: http://dx.doi.org/10.1080/01691864.2015.1122553

[13] P. Liljeback, K. Y. Pettersen, and O. Stavdahl, "A snake robot with a contact force measurement system for obstacle-aided locomotion," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, Conference Proceedings, pp. 683–690. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5509839

[14] P. Liljebäck and R. Mills, "Eelume: A flexible and subsea resident imr vehicle," in *OCEANS 2017 - Aberdeen*, Conference Proceedings, pp. 1–4.

[15] E. S. Intervention, "Eelume," https://eelume.com/.

[16] C. Wright, A. Johnson, A. Peck, Z. McCord, A. Naaktgeboren, P. Gianfortoni, M. Gonzalez-Rivero, R. Hatton, and H. Choset, "Design of a modular snake robot," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, Conference Proceedings, pp. 2609–2614. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4399617

[17] K. Lipkin, I. Brown, A. Peck, H. Choset, J. Rembisz, P. Gianfortoni, and A. Naaktgeboren, "Differentiable and piecewise differentiable gaits for snake robots," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2007, October 29, 2007 - November 2, 2007*, ser. IEEE International Conference on Intelligent Robots and Systems. Institute of Electrical and Electronics Engineers Inc., Conference Proceedings, pp. 1864–1869. [Online]. Available: http://dx.doi.org/10.1109/IROS.2007.4399638http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4399638

[18] C. Wright, A. Buchan, B. Brown, J. Geist, M. Schwerin, D. Rollinson, M. Tesch, and H. Choset, "Design and architecture of the unified modular snake robot," in *2012 IEEE International Conference on Robotics and Automation, ICRA 2012*, ser. Proceedings - IEEE International Conference on Robotics and Automation. Institute of Electrical and Electronics Engineers Inc., Conference Proceedings, pp. 4347–4354. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2012.6225255http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6225255

[19] D. Rollinson, Y. Bilgen, B. Brown, F. Enner, S. Ford, C. Layton, J. Rembisz, M. Schwerin, A. Willig, P. Velagapudi, and H. Choset, "Design and architecture of a series elastic snake robot," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, Conference Proceedings, pp. 4630–4636. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6943219

[20] "The robot operating system (ros)," http://www.ros.org/.

[21] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[22] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[23] ——, "A new extension of the kalman filter to nonlinear systems," in *Int. symp. aerospace/defense sensing, simul. and controls*, vol. 3. Orlando, FL, Conference Proceedings, pp. 182–193.

[24] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2006.

[25] M. St-Pierre and D. Gingras, "Comparison between the unscented kalman filter and the extended kalman filter for the position estimation module of an integrated navigation information system," in *IEEE Intelligent Vehicles Symposium, 2004*, Conference Proceedings, pp. 831–835.

[26] J. Gaebler, S. Hur-Diaz, and R. Carpenter, "Comparison of sigma-point and extended kalman filters on a realistic orbit determination scenario," *The Journal of the Astronautical Sciences*, vol. 59, no. 1, pp. 301–307, 2012. [Online]. Available: https://doi.org/10.1007/s40295-013-0019-0

[27] R. Van Der Merwe, E. A. Wan, and S. Julier, "Sigma-point kalman filters for nonlinear estimation and sensor-fusion: Applications to integrated navigation," in *Proceedings of the AIAA Guidance, Navigation & Control Conference*, Conference Proceedings, pp. 16–19.

[28] S. J. Julier, "The spherical simplex unscented transformation," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 3, Conference Proceedings, pp. 2430–2434 vol.3.

[29] T. Xiaojun, Z. Xiaobei, and Z. Xubin, "The square-root spherical simplex unscented kalman filter for state and parameter estimation," in *2008 9th International Conference on Signal Processing*, Conference Proceedings, pp. 260–263.

[30] J. G. C. Lozano, L. R. G. Carrillo, A. Dzul, and R. Lozano, "Spherical simplex sigma-point kalman filters: A comparison in the inertial navigation of a terrestrial vehicle," in *2008 American Control Conference*, Conference Proceedings, pp. 3536–3541.

[31] D. Rollinson, A. Buchan, and H. Choset, "State estimation for snake robots," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, Conference Proceedings, pp. 1075–1080. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6095052

[32] D. Rollinson, H. Choset, and S. Tully, "Robust state estimation with redundant proprioceptive sensors," in *ASME 2013 Dynamic Systems and Control Conference, DSCC 2013, October 21, 2013 - October 23, 2013*, ser. ASME 2013 Dynamic Systems and Control Conference, DSCC 2013, vol. 3. American Society of Mechanical Engineers (ASME), Conference Proceedings, p. Dynamic Systems and Control Division. [Online]. Available: http://dx.doi.org/10.1115/DSCC2013-3873http://proceedings.asmedigitalcollection.asme.org/proceeding.aspx?articleid=1841358

[33] D. Rollinson and H. Choset, "Gait-based compliant control for snake robots," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, Conference Proceedings, pp. 5138–5143. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6631311

[34] Z. Zhang, J. Shang, C. Seneci, and G. Z. Yang, "Snake robot shape sensing using micro-inertial sensors," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, Conference Proceedings, pp.

831–836. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6696447

[35] Z. Y. Bayraktaroglu, "Snake-like locomotion: Experimentations with a biologically inspired wheel-less snake robot," *Mechanism and Machine Theory*, vol. 44, no. 3, pp. 591–602, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.mechmachtheory.2008.08.009http://www.sciencedirect.com/science/article/pii/S0094114X08001791

[36] P. Liljeback, K. Y. Pettersen, O. Stavdahl, and J. T. Gravdahl, "Hybrid modelling and control of obstacle-aided snake robot locomotion," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 781–799, 2010. [Online]. Available: http://dx.doi.org/10.1109/TRO.2010.2056211http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5535144

[37] ——, "Experimental investigation of obstacle-aided locomotion with a snake robot," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 792–800, 2011. [Online]. Available: http://dx.doi.org/10.1109/TRO.2011.2134150http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5759101

[38] S. Hirose, *Biologically Inspired Robots: Snake-Like Locomotors and Manipulators*. Oxford: Oxford University Press, 1993.

[39] S. O. Madgwick, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays," 2010.

[40] N. Ammann, A. Derksen, and C. Heck, "A novel magnetometer-accelerometer calibration based on a least squares approach," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, Conference Proceedings, pp. 577–585.

[41] T. Kugelstadt, "Extending the spi bus for long-distance communication," Texas Instruments, Tech. Rep., 2011.

[42] S. Merkel, "Extending i2c communication distance with the ds28e17," Maxim Integrated Products, Tech. Rep., 2015.

[43] T. Kugelstadt, "The rs-485 design guide," Texas Instruments, Tech. Rep., 2008.

[44] C. Kinnaird, "Use receiver equalization to extend rs-485 data communications," Texas Instruments, Tech. Rep., 2004.

[45] A. N. 643, "Pre-emphasis improves rs-485 communications," Maxim Integrated Products, Tech. Rep., 2001.

[46] S. Corrigan, "Introduction to the controller area network (can)," Texas Instruments, Tech. Rep., 2002.

[47] "Can – a short introduction," https://www.ixxat.com/technologies/fieldbuses/can.

[48] D. Automation, "Rs485 network for modbus," https://www.drago-automation.de/frequently-asked-questions/items/rs485-network-for-modbus.html.

[49] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, "Continuous clock synchronization in wireless real-time applications," in *Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000*, Conference Proceedings, pp. 125–132.

[50] I.-K. Rhee, J. Lee, J. Kim, E. Serpedin, and Y.-C. Wu, "Clock synchronization in wireless sensor networks: An overview," *Sensors*, vol. 9, no. 1, pp. 56–85, 2009.

[51] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM, Conference Proceedings, pp. 138–149.

[52] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 147–163, 2002.

[53] M. Ferguson, "Rosserial," http://wiki.ros.org/rosseriall.

[54] D. R. A. B. H. Choset, "Virtual chassis for snake robot: Definition and applications," *Advanced Robotics*, vol. Oct, pp. 1–22, 2012.

[55] C. Gilani, X. Chen, C. Pretty, and C. Koike, "Visualisation of the motion trajectory for rolling motion of snake robots using virtual chassis and simplified kinematics motion model," *IFAC-PapersOnLine*, vol. 49, no. 32, pp. 228–233, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896316328919

[56] R. Bro, E. Acar, and T. G. Kolda, "Resolving the sign ambiguity in the singular value decomposition," *Journal of Chemometrics*, vol. 22, no. 2, pp. 135–140, 2008.

[57] S. J. Julier, "The scaled unscented transformation," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, vol. 6, Conference Proceedings, pp. 4555–4559 vol.6.

[58] S. Eun-Hwan and N. El-Sheimy, "An unscented kalman filter for in-motion alignment of low-cost imus," in *PLANS 2004. Position Location and Navigation Symposium (IEEE Cat. No.04CH37556)*, Conference Proceedings, pp. 273–279.

[59] R. Van der Merwe and E. A. Wan, "The square-root unscented kalman filter for state and parameter-estimation," in *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, vol. 6, Conference Proceedings, pp. 3461–3464 vol.6.

[60] G. G. Slabaugh, "Computing euler angles from a rotation matrix," 1999.

[61] E. Kraft, "A quaternion-based unscented kalman filter for orientation tracking," in *Proceedings of the Sixth International Conference of Information Fusion*, vol. 1, Conference Proceedings, pp. 47–54.

[62] N. Enayati, "Quaternion based unscented kalman filter for robust motion tracking in neurosurgery," dissertation, Politecnico Di Milano, 2013.

[63] N. Gammon, "Rs485 non-blocking arduino library," https://www.gammon.com.au/forum/?id=11428.

[64] A. Arkhipenko, "Task scheduler," http://playground.arduino.cc/Code/TaskScheduler, 2015, accessed: 2016-8-20.

[65] M. Herb, "Kalman filter library," https://github.com/mherb/kalman.

Appendices

Appendix A

IMU Component Properties

Table A.1: Acceleration Properties.

Model	**FSR (g)**	**Sensitivity ±16 (mg/LSB)**
LSM6DS33	$\pm 2, \pm 4, \pm 8, \pm 16$	0.488
LSM303D	$\pm 2, \pm 4, \pm 6, \pm 8, \pm 16$	0.732
LSM9DS0	$\pm 2, \pm 4, \pm 6, \pm 8, \pm 16$	0.732
LSM9DS1	$\pm 2, \pm 4, \pm 8, \pm 16$	0.732
MPU9150	$\pm 2, \pm 4, \pm 8, \pm 16$	0.488
MPU9255	$\pm 2, \pm 4, \pm 8, \pm 16$	0.488

Table A.2: Gyroscope Properties.

Model	**FSR (dps)**	**Sensitivity ±500 (mdps/LSB)**
LSM6DS33	$\pm 125, \pm 245, \pm 500, \pm 1000, \pm 2000$	17.50
L3GD20H	$\pm 245, \pm 500, \pm 2000$	17.50
LSM9DS0	$\pm 245, \pm 500, \pm 2000$	17.50
LSM9DS1	$\pm 245, \pm 500, \pm 2000$	17.50
MPU9150	$\pm 250, \pm 500, \pm 1000, \pm 2000$	15.27
MPU9255	$\pm 250, \pm 500, \pm 1000, \pm 2000$	15.27

Table A.3: Magnetometer Properties.

Model	**FSR (gauss)**	**Sensitivity** ± 12 **(mgauss/LSB)**
LIS3MDL	$\pm 4, \pm 8, \pm 12, \pm 16$	0.438
LSM303D	$\pm 2, \pm 4, \pm 8, \pm 12$	0.479
LSM9DS0	$\pm 2, \pm 4, \pm 8, \pm 12$	0.48
LSM9DS1	$\pm 4, \pm 8, \pm 12, \pm 16$	0.43
MPU9150	± 12	3
MPU9255	± 48	6

Appendix B

Base Station PCB

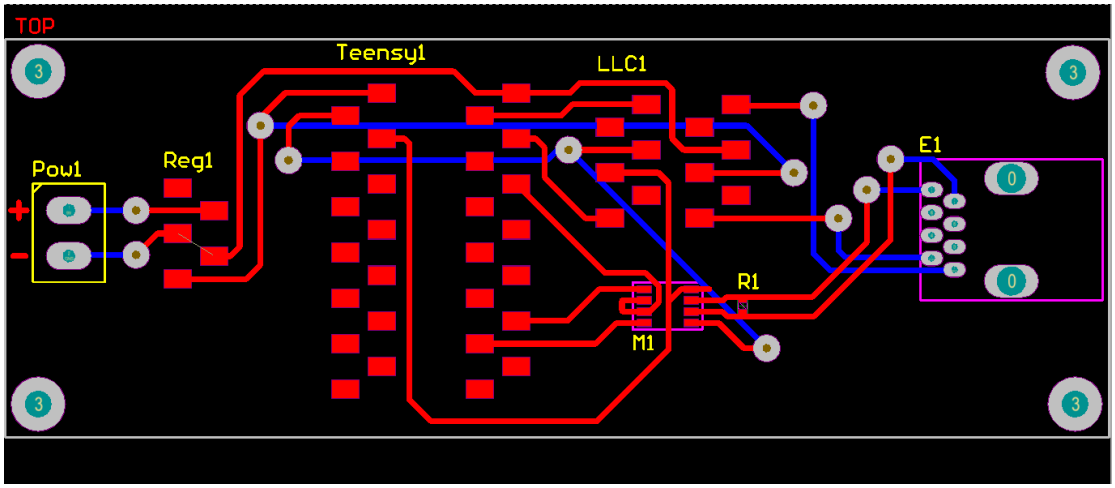


Figure B.1: PCB of the base station.

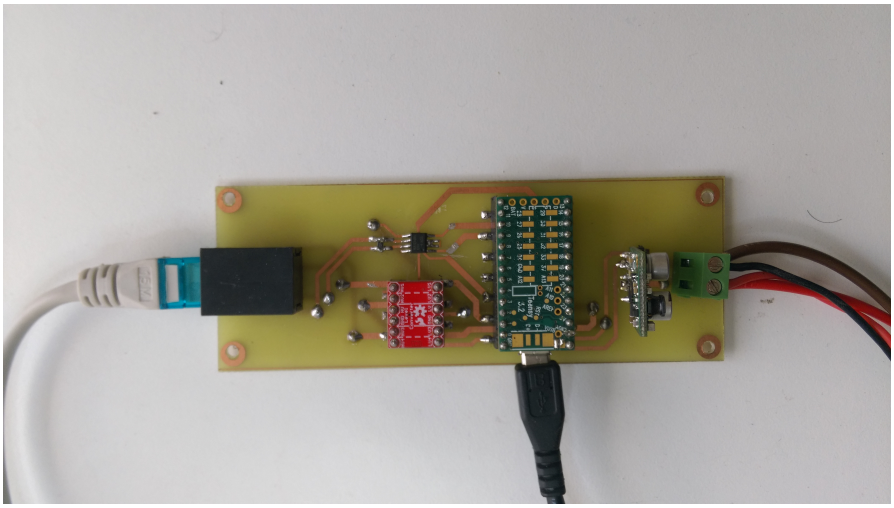


Figure B.2: Photo of the base station.

Appendix C

UKF Scaling Factors

Table C.1: Scaling factors for the rolling gait.

Parameter	**Scaling Value**
A	1
$\dot{A}$	1
V	0.05
$\dot{V}$	0.5
$\boldsymbol{e}$	1
$\boldsymbol{q}$	0.2
$\boldsymbol{\omega}$	0.1

Table C.2: Scaling factors for the rotating gait.

Parameter	**Scaling Value**
A_y	1
$\dot{A}_y$	1
A_p	1
$\dot{A}_p$	1
V	0.05
$\dot{V}$	0.5
ψ_y	0.05
$\dot{\psi}_y$	0.5
ψ_p	0.05
$\dot{\psi}_p$	0.5
$\boldsymbol{e}$	1
$\boldsymbol{q}$	0.2
$\boldsymbol{\omega}$	0.1

Table C.3: Scaling factors for the joint angles model.

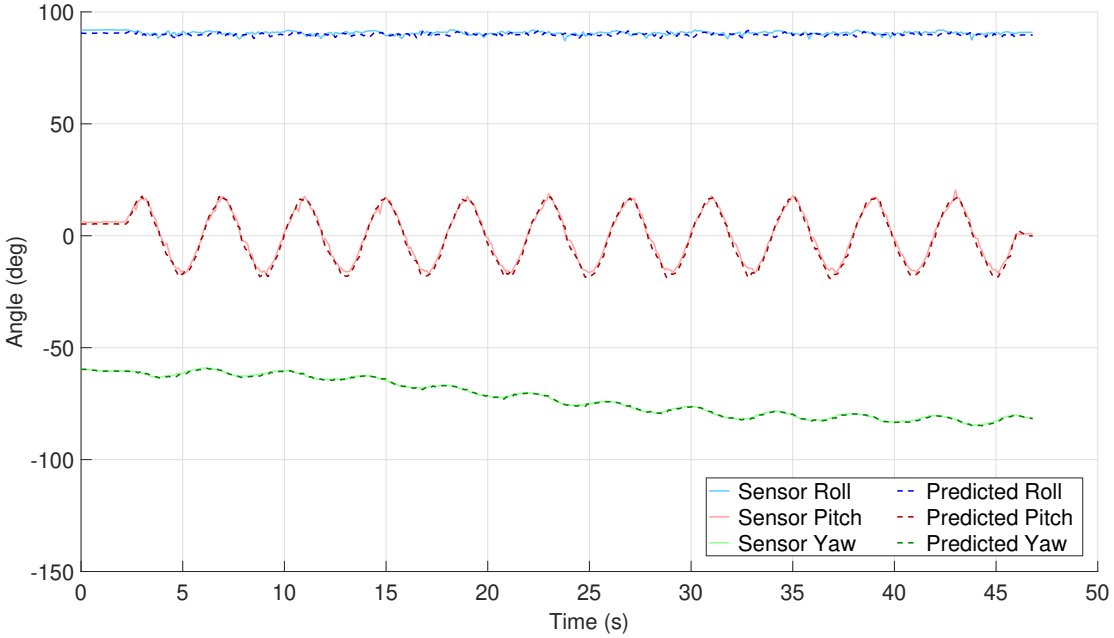
Parameter	**Scaling Value**
θ	1
$\dot{\theta}$	2
$\ddot{\theta}$	4
$\boldsymbol{q}$	0.2
$\boldsymbol{\omega}$	0.1

Appendix D

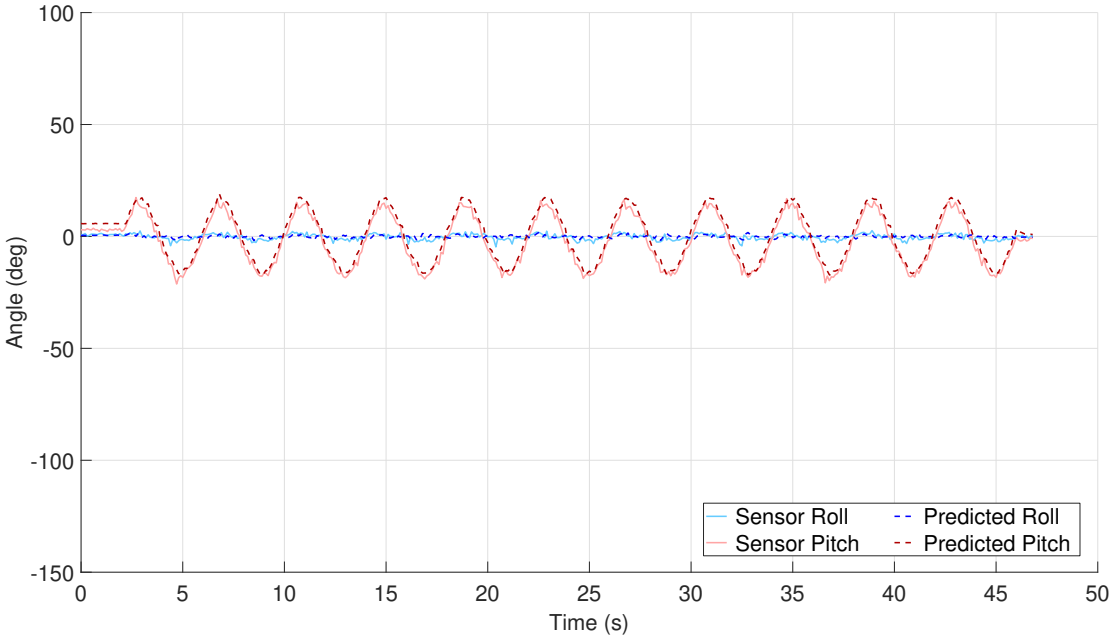
Internal Test Data

Table D.1: Internal test data mean error for modules 0 and 9.

Gait	**Module**	**Roll Error (Degrees)**	**Pitch Error (Degrees)**	**Yaw Error (Degrees)**
Linear Progression	0	0.94	1.42	0.28
Linear Progression	9	1.02	2.47	–
Turning	0	1.58	1.10	0.42
Turning	9	1.45	2.45	–
Rolling	0	2.07	1.72	3.00
Rolling	9	2.31	2.04	–
Rotating	0	3.78	1.88	3.35
Rotating	9	4.24	2.76	–
Angles Linear Progression	0	0.99	1.57	0.30
Angles Linear Progression	9	1.06	2.69	–
Angles Turning	0	2.49	1.16	0.74
Angles Turning	9	2.40	2.69	–
Angles Rolling	0	2.43	1.75	2.93
Angles Rolling	9	2.17	2.18	–
Angles Rotating	0	4.30	1.82	3.57
Angles Rotating	9	4.62	2.75	–

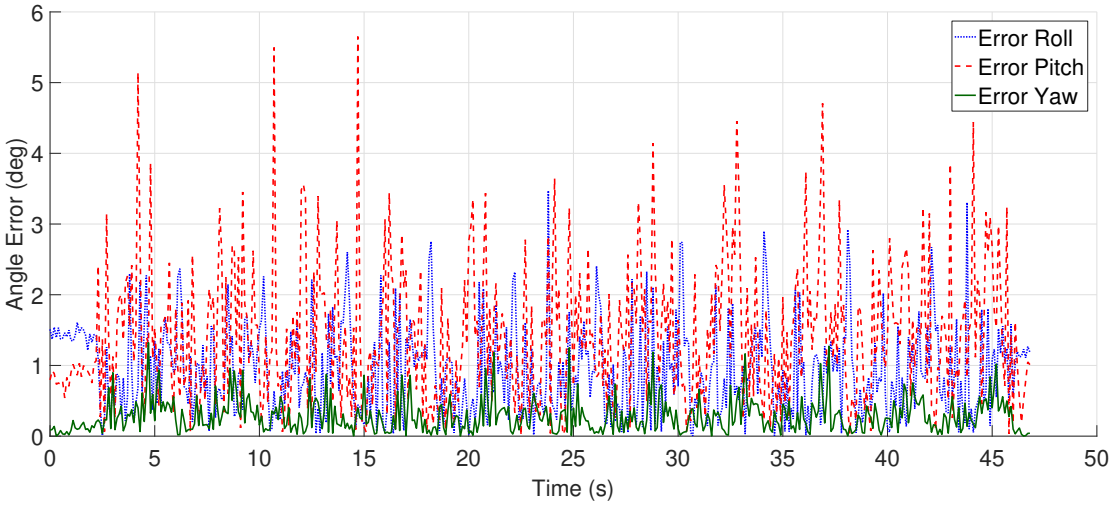


(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the linear progression gait using the gait parameters model.

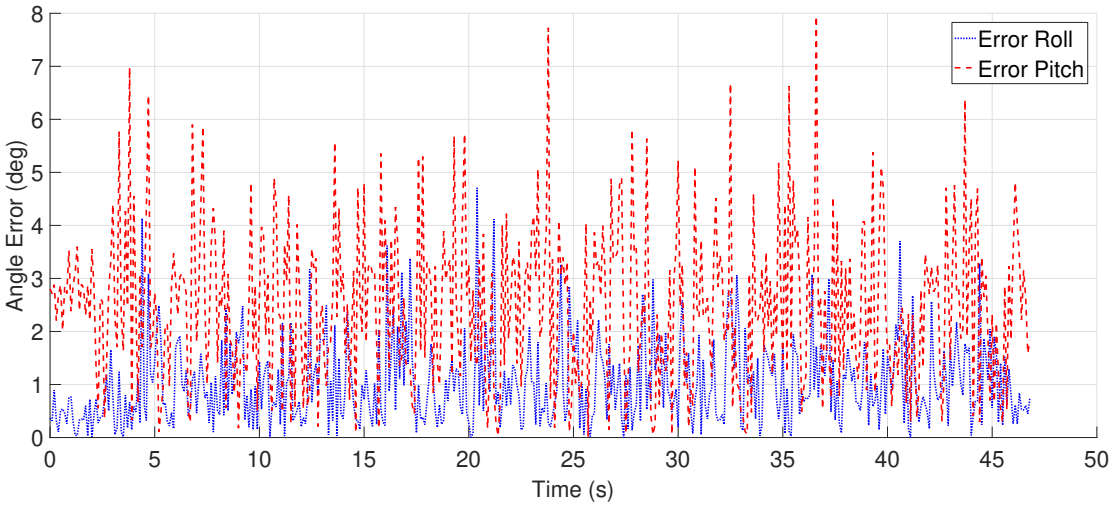


(b) Comparison between the predicted Euler angles and the sensor data for module 9 while performing the linear progression gait using the gait parameters model.

Figure D.1: Predicted pose of modules 0 and 9 with the SR-SSUKF while using the linear progression gait with the gait parameters model.

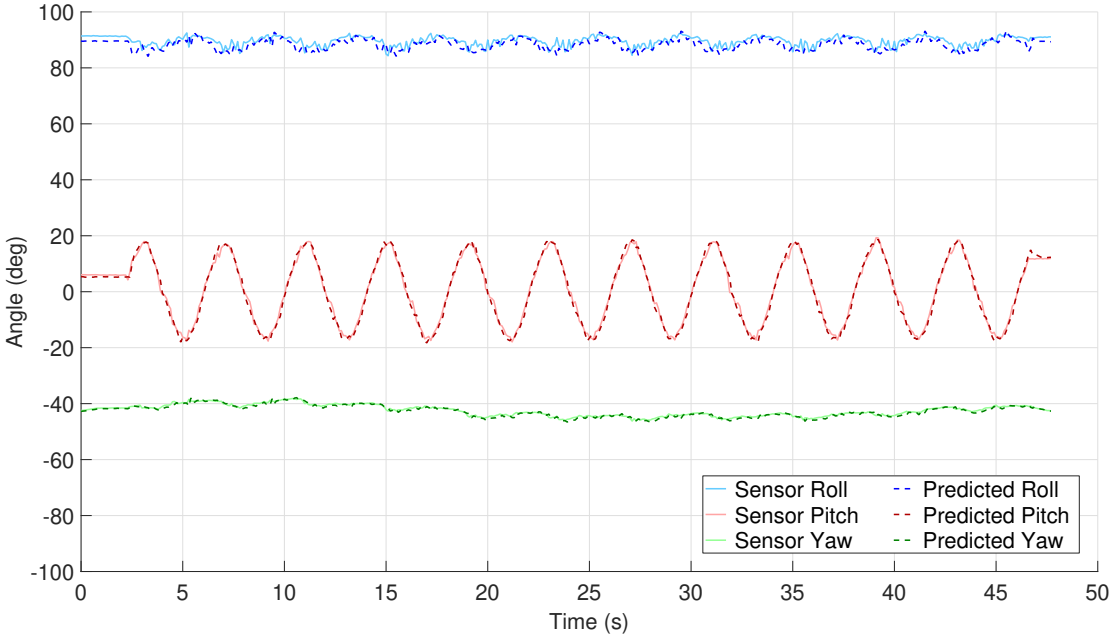


(a) Prediction error of module 0 using the linear progression gait with the gait parameters model.

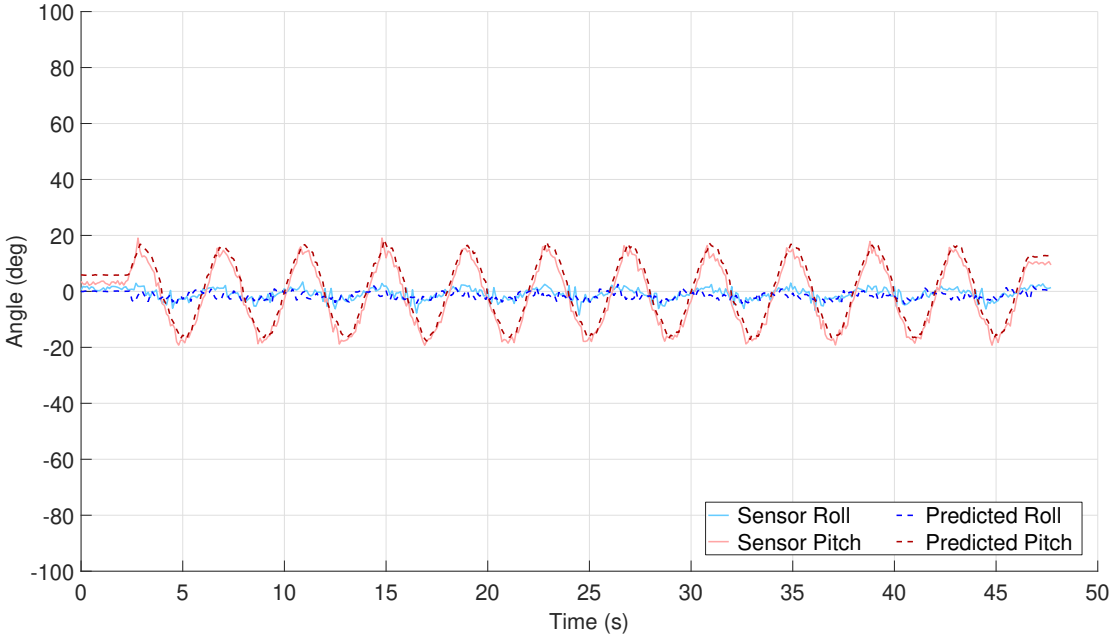


(b) Prediction error of module 9 using the linear progression gait with the gait parameters model.

Figure D.2: Predicted error of modules 0 and 9 with the SR-SSUKF while using the linear progression gait with the gait parameters model.

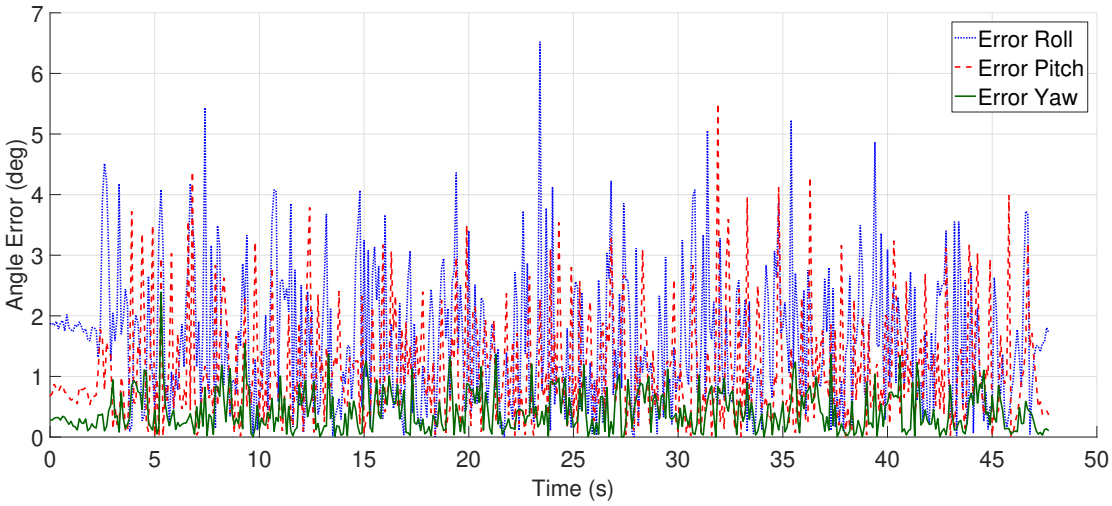


(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the turning gait using the gait parameters model.

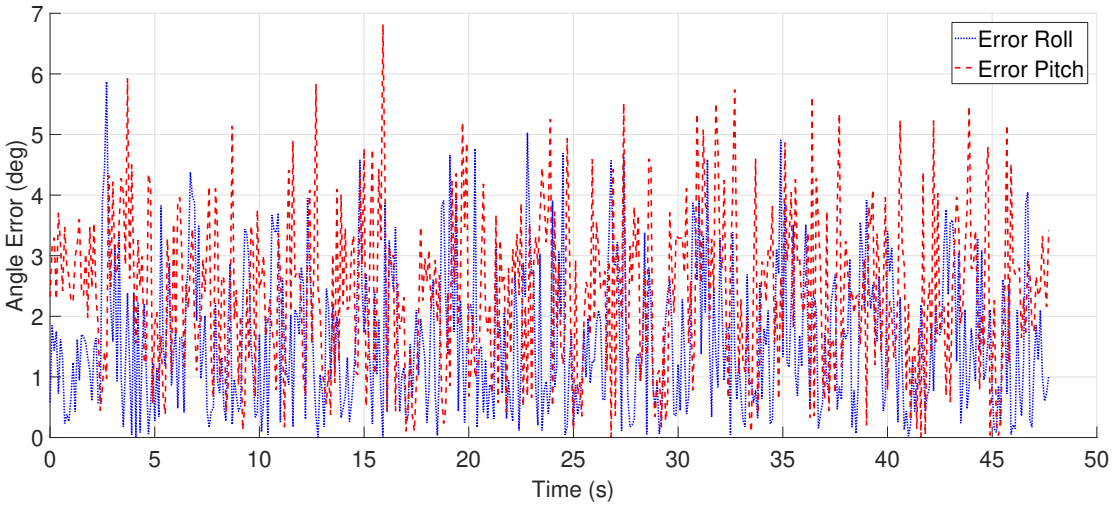


(b) Comparison between the predicted Euler angles and the sensor data for module 9 while performing the turning gait using the gait parameters model.

Figure D.3: Predicted pose of modules 0 and 9 with the SR-SSUKF while using the turning gait with the gait parameters model.

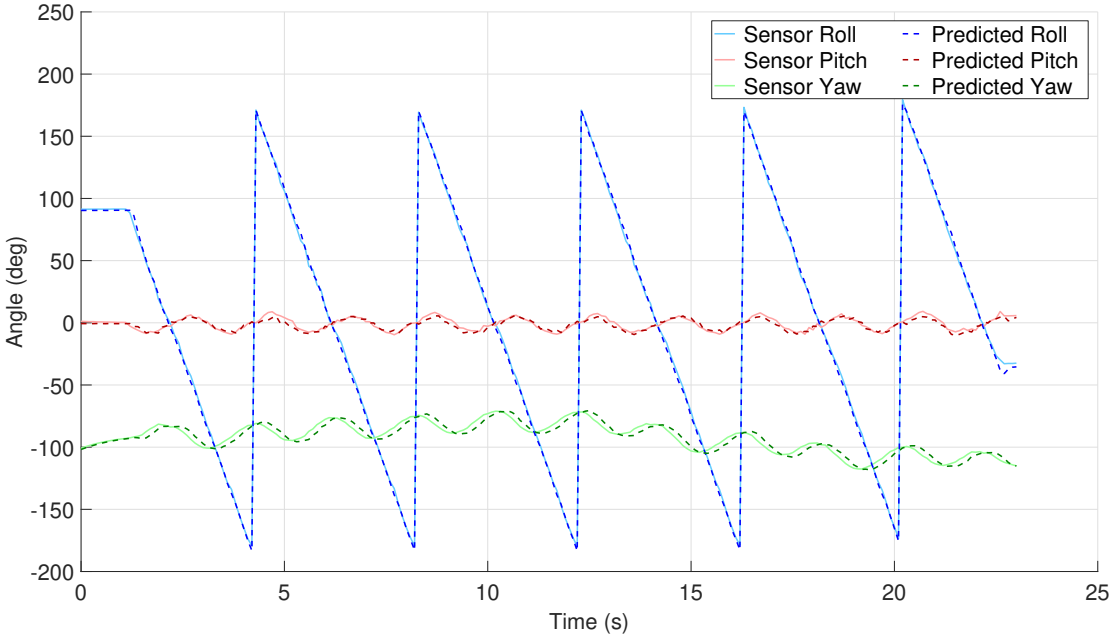


(a) Prediction error of module 0 using the turning gait with the gait parameters model.

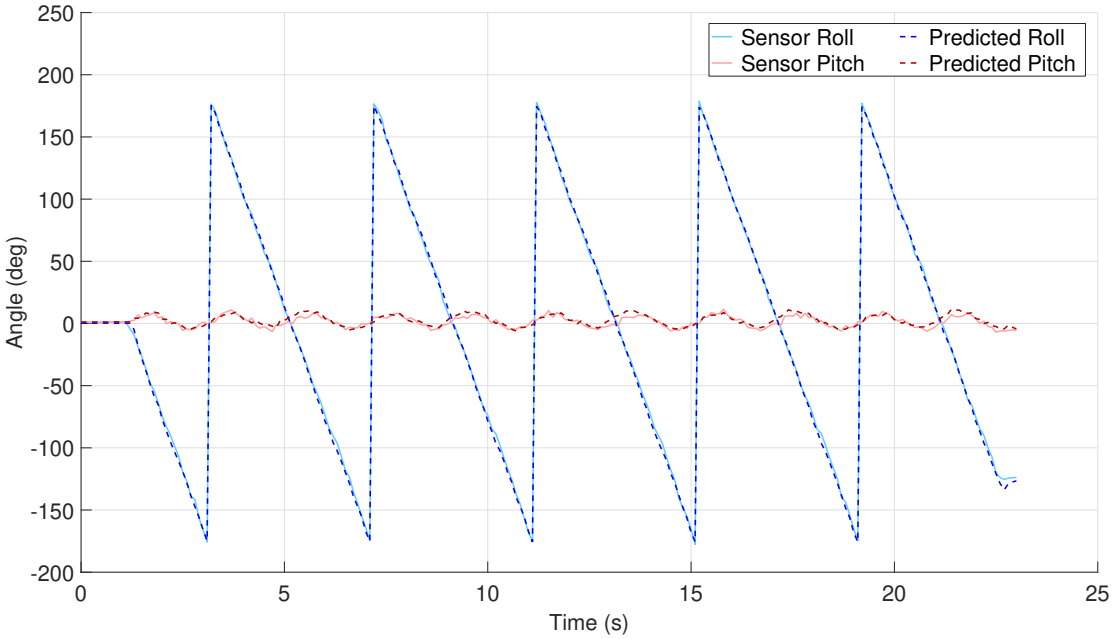


(b) Prediction error of module 9 using the turning gait with the gait parameters model.

Figure D.4: Predicted error of modules 0 and 9 with the SR-SSUKF while using the turning gait with the gait parameters model.

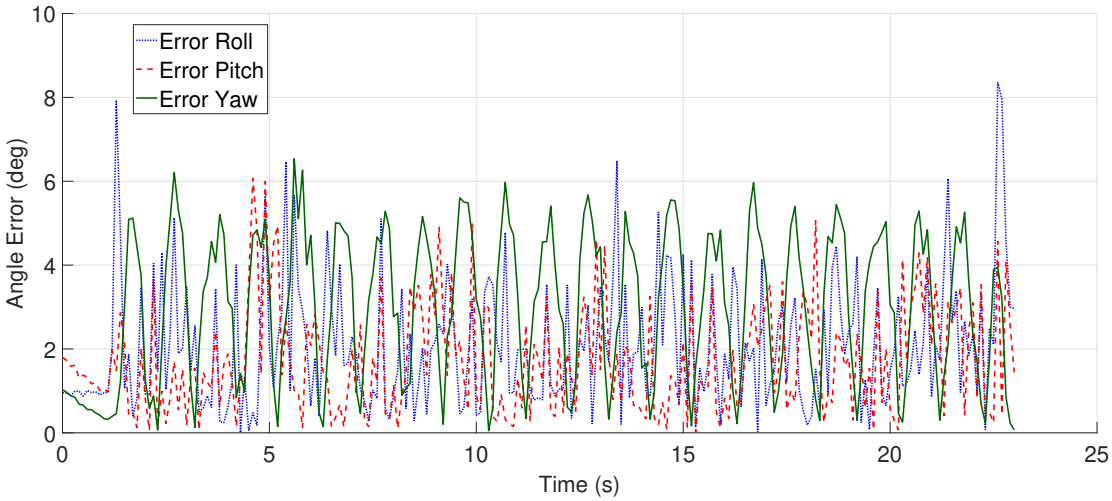


(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the rolling gait using the gait parameters model.

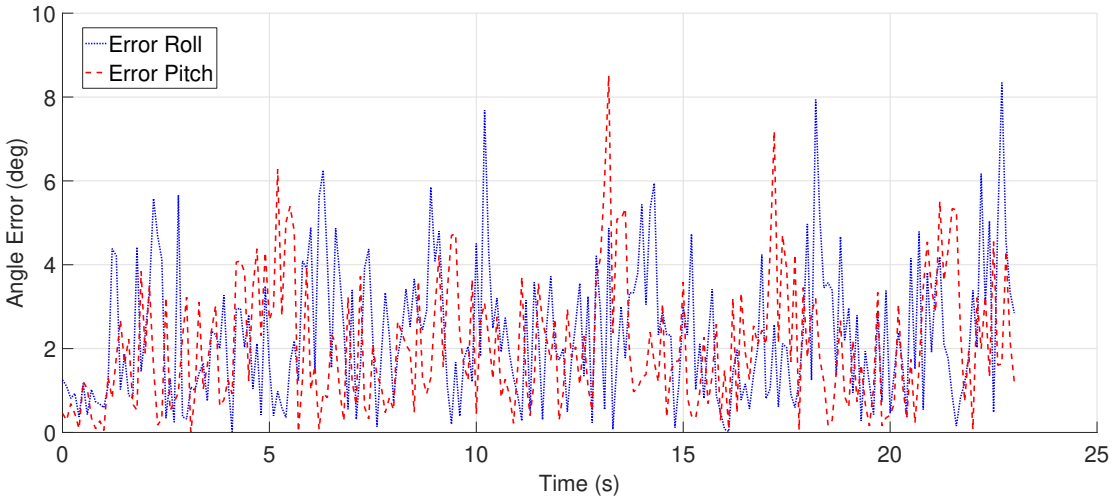


(b) Comparison between the predicted Euler angles and the sensor data for module 9 while performing the rolling gait using the gait parameters model.

Figure D.5: Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the gait parameters model.

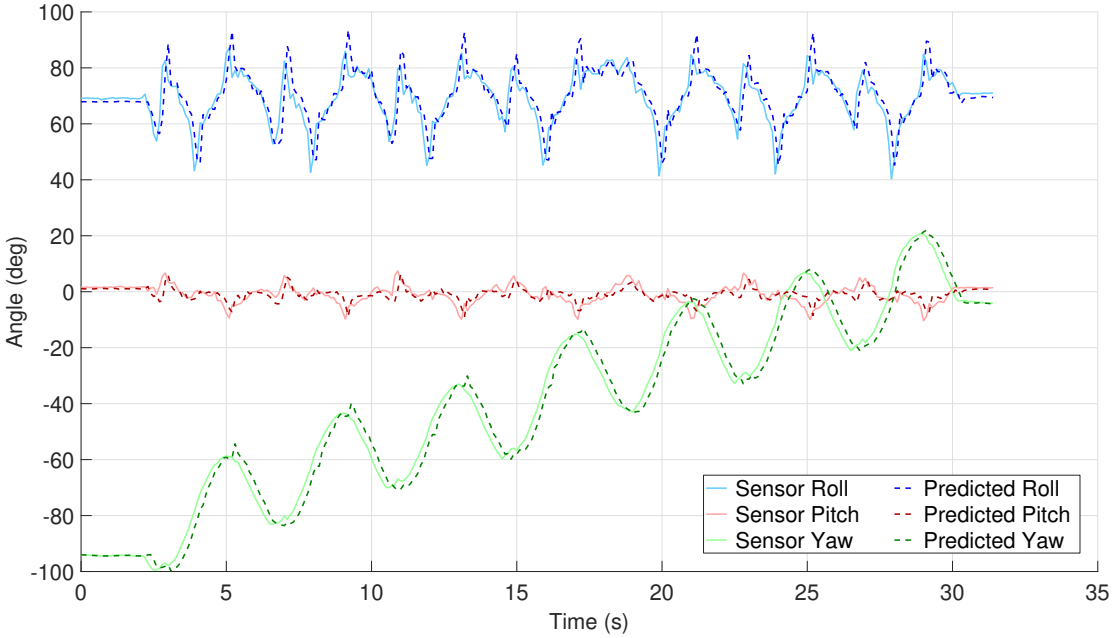


(a) Prediction error of module 0 using the rolling gait with the gait parameters model.

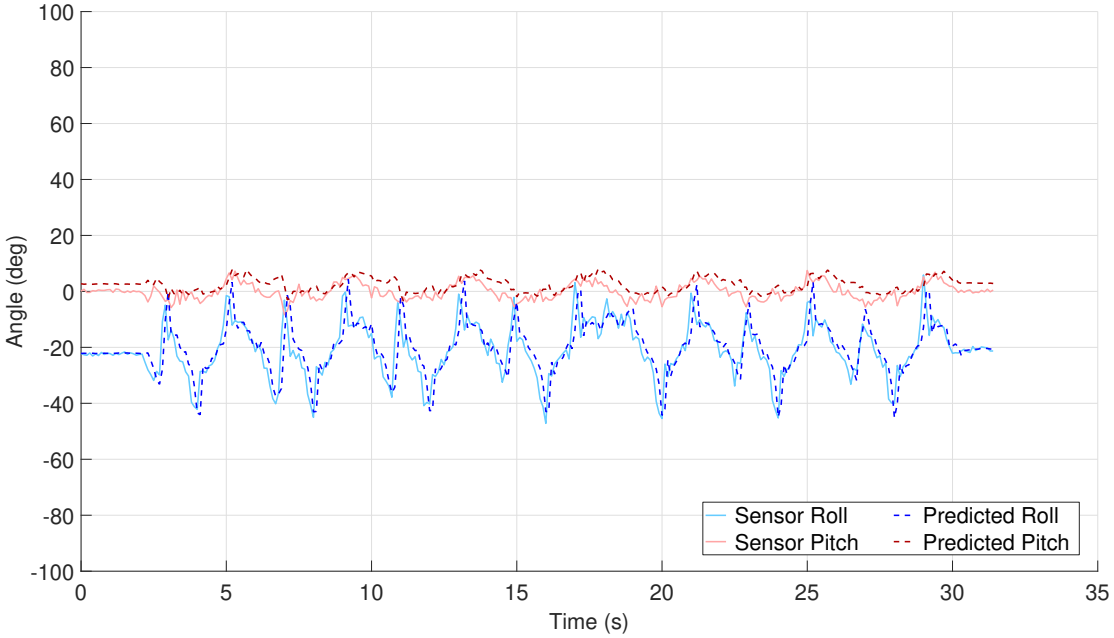


(b) Prediction error of module 9 using the rolling gait with the gait parameters model.

Figure D.6: Predicted error of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the gait parameters model.

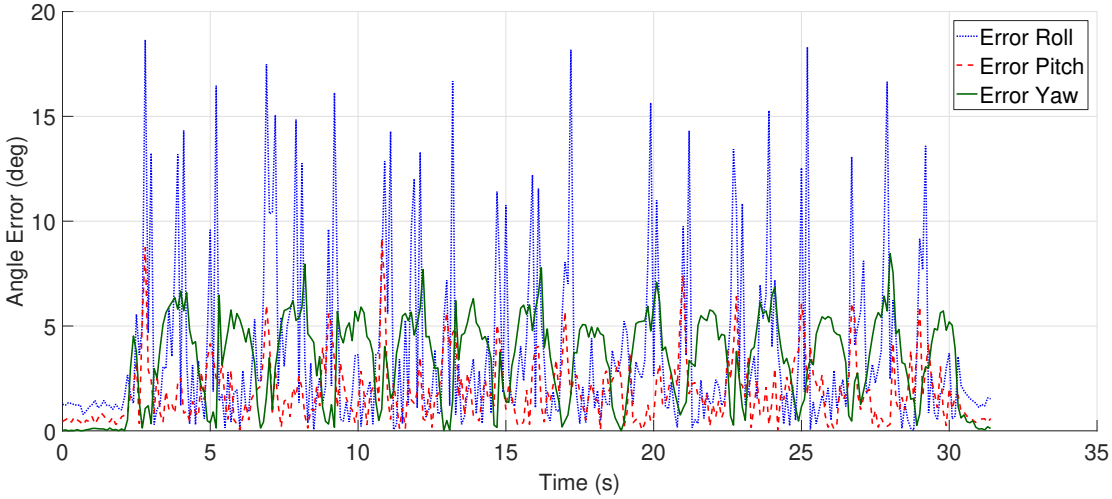


(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the rotating gait using the gait parameters model.

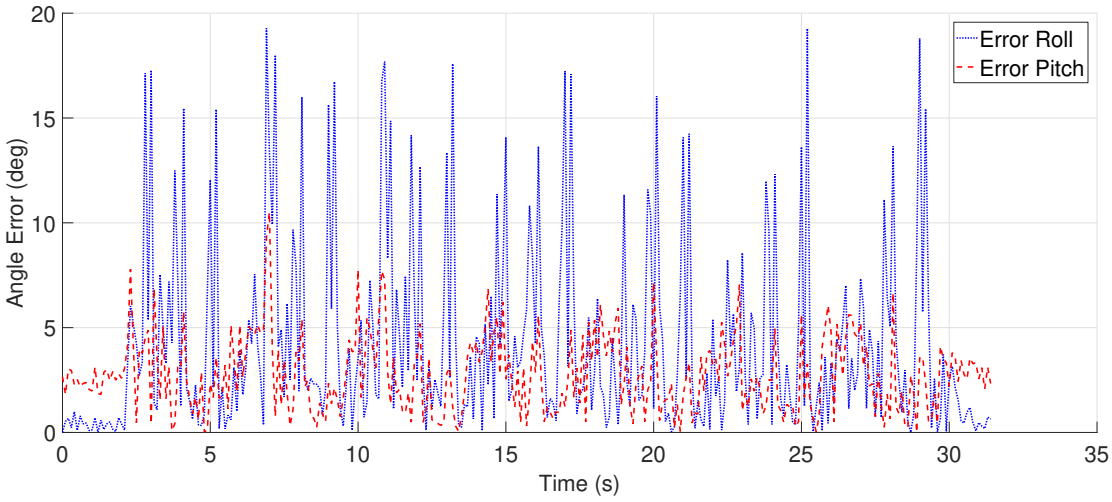


(b) Comparison between the predicted Euler angles and the sensor data for module 9 while performing the rotating gait using the gait parameters model.

Figure D.7: Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rotating gait with the gait parameters model.

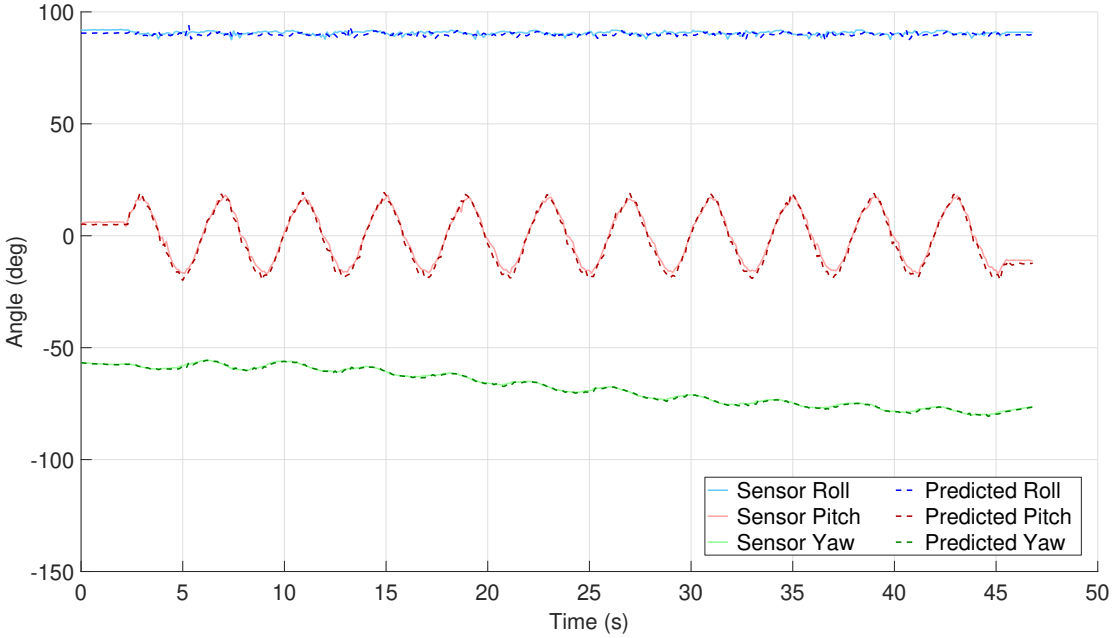


(a) Prediction error of module 0 using the rotating gait with the gait parameters model.

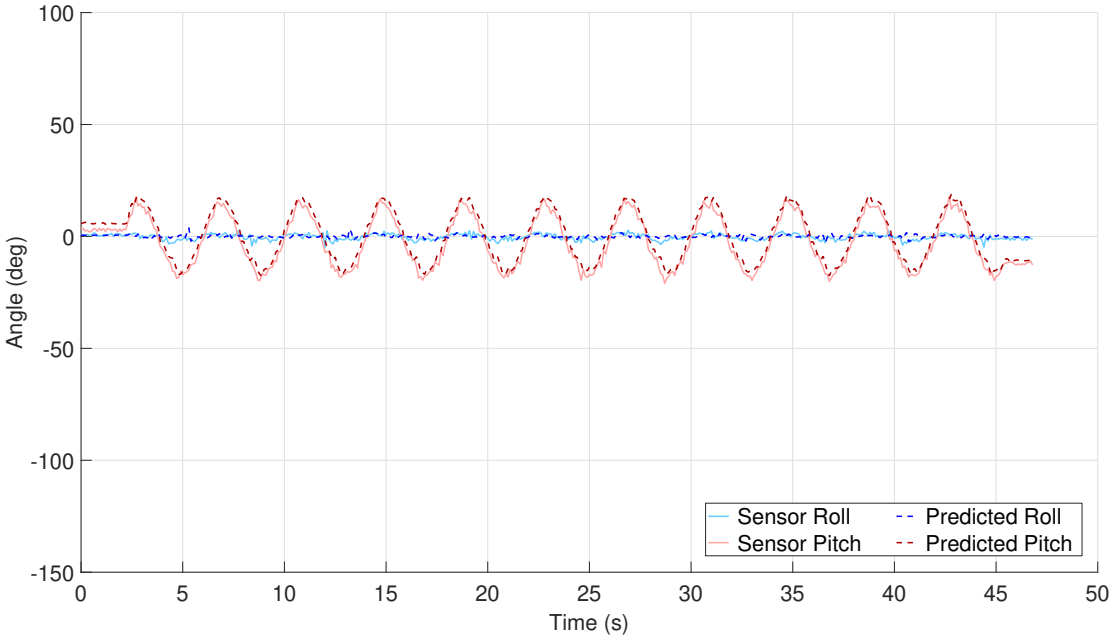


(b) Prediction error of module 9 using the rotating gait with the gait parameters model.

Figure D.8: Predicted error of modules 0 and 9 with the SR-SSUKF while using the rotating gait with the gait parameters model.

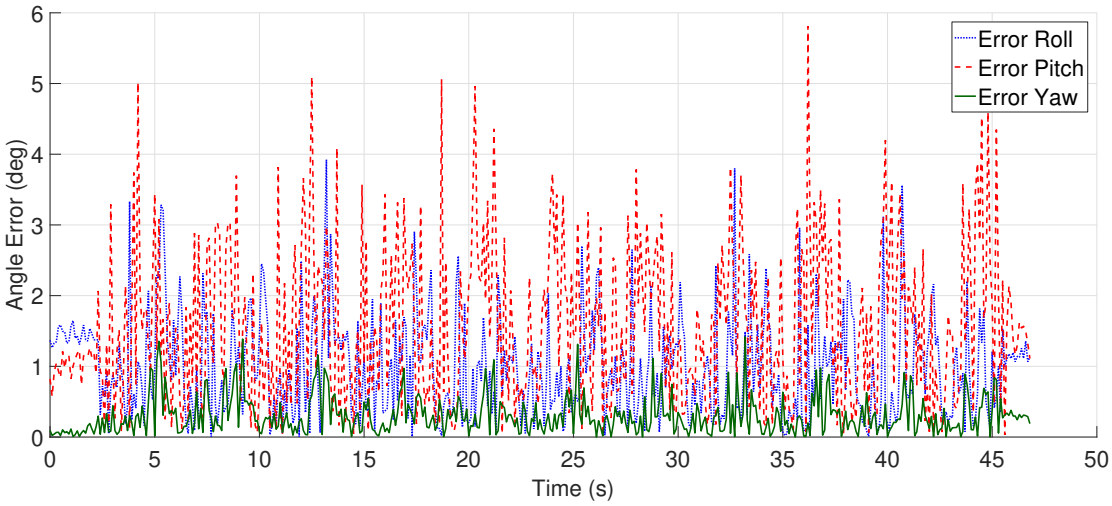


(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the linear progression gait using the joint angles model.

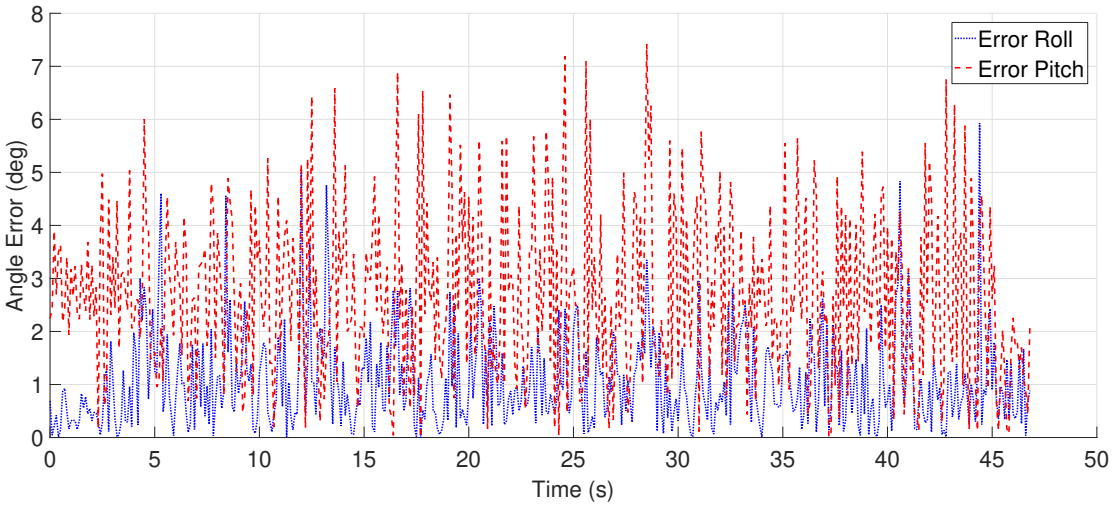


(b) Comparison between the predicted Euler angles and the sensor data for module 9 while performing the linear progression gait using the joint angles model.

Figure D.9: Predicted pose of modules 0 and 9 with the SR-SSUKF while using the linear progression gait with the joint angles model.

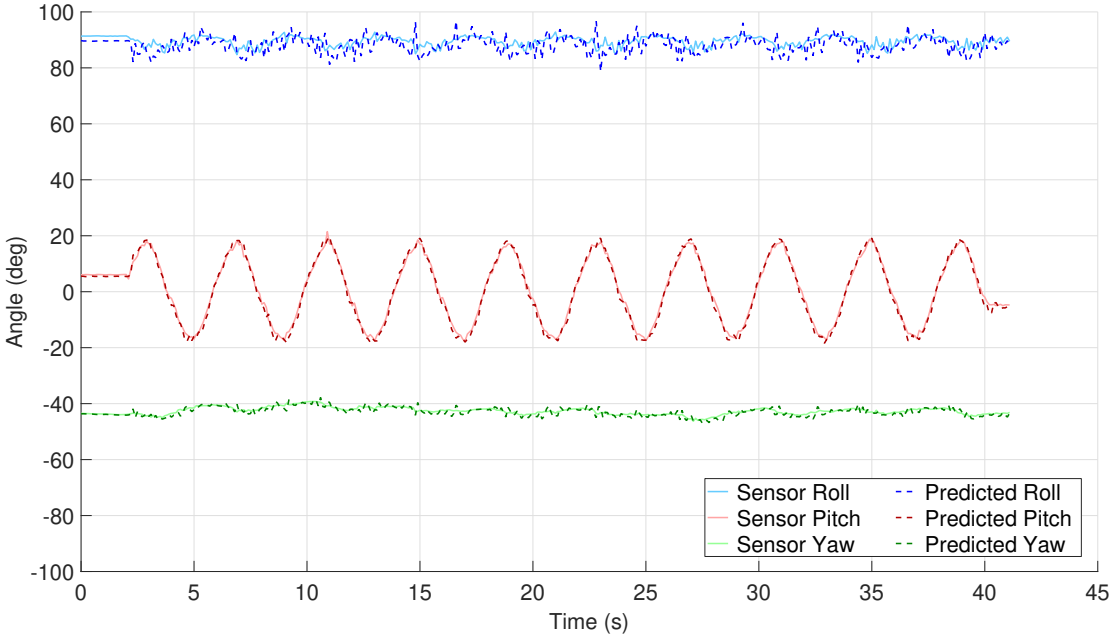


(a) Prediction error of module 0 using the linear progression gait with the joint angles model.

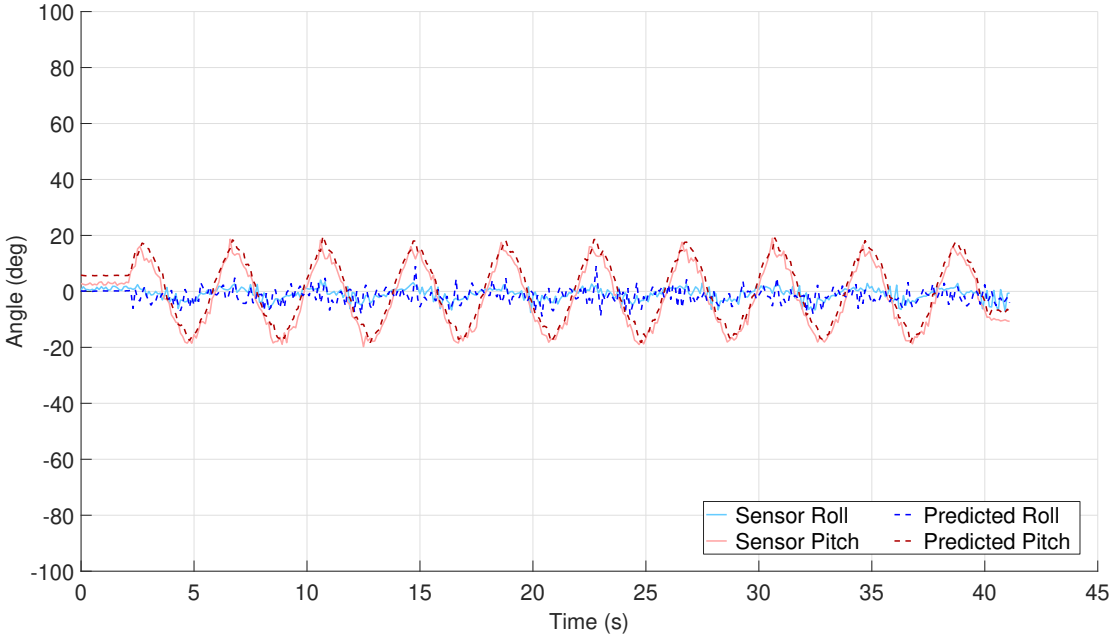


(b) Prediction error of module 9 using the linear progression gait with the joint angles model.

Figure D.10: Predicted error of modules 0 and 9 with the SR-SSUKF while using the linear progression gait with the joint angles model.

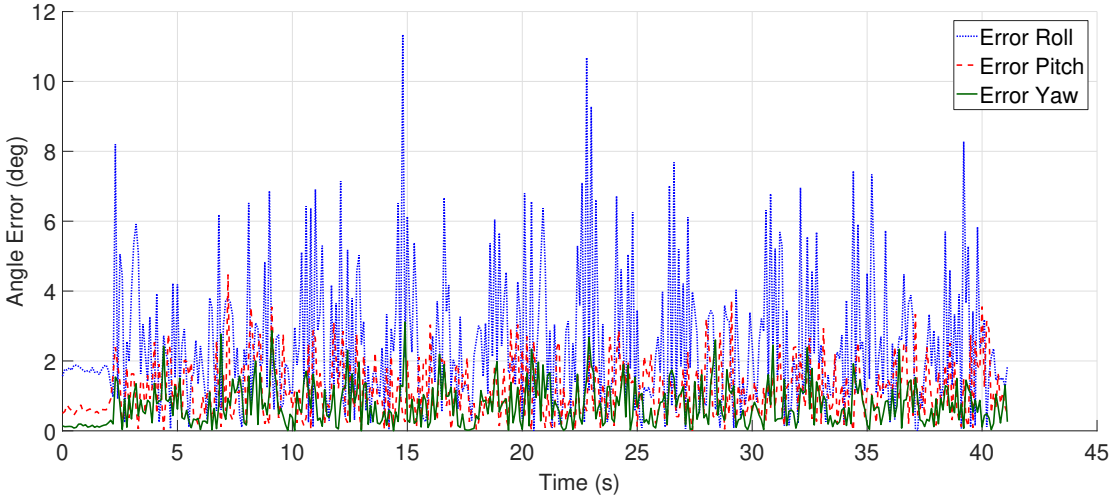


(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the turning gait using the joint angles model.

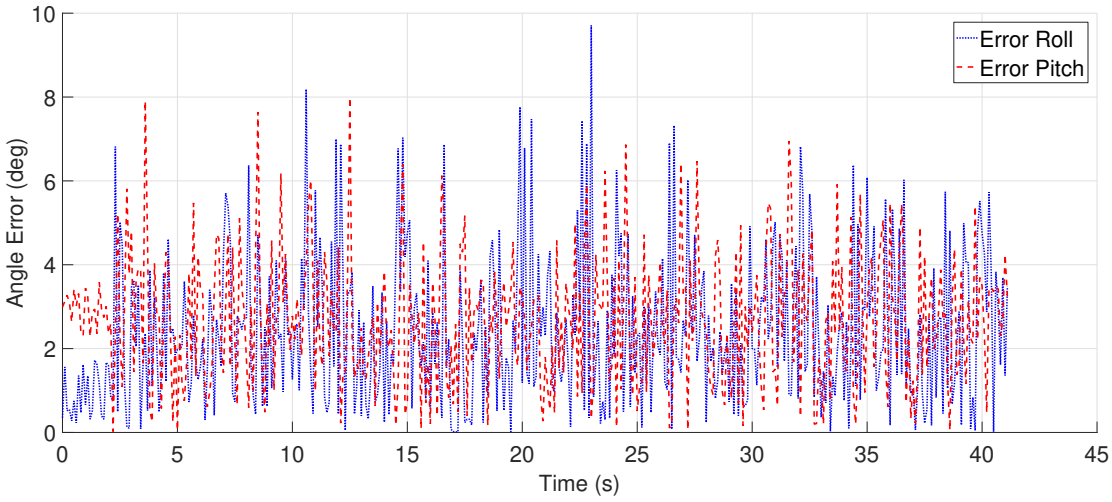


(b) Comparison between the predicted Euler angles and the sensor data for module 9 while performing the turning gait using the joint angles model.

Figure D.11: Predicted pose of modules 0 and 9 with the SR-SSUKF while using the turning gait with the joint angles model.

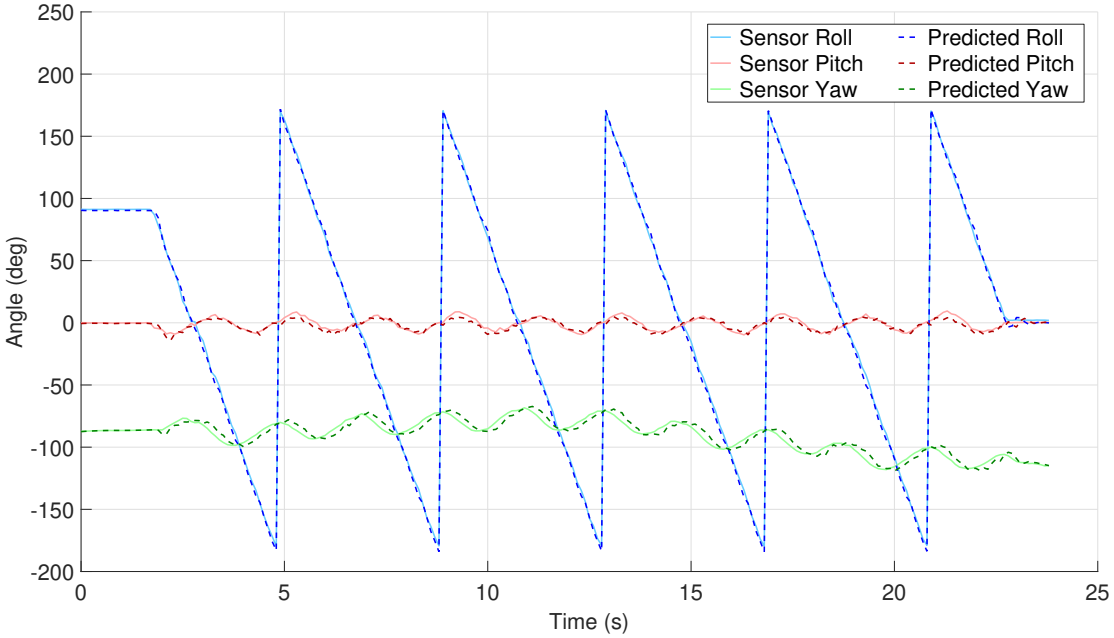


(a) Prediction error of module 0 using the turning gait with the joint angles model.

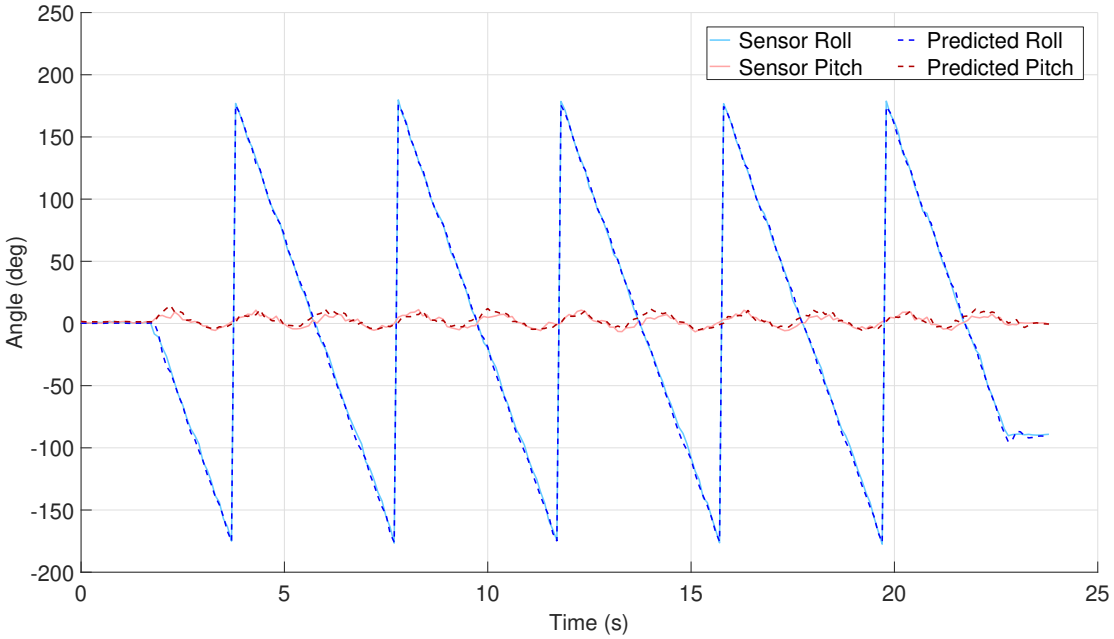


(b) Prediction error of module 9 using the turning gait with the joint angles model.

Figure D.12: Predicted error of modules 0 and 9 with the SR-SSUKF while using the turning gait with the joint angles model.

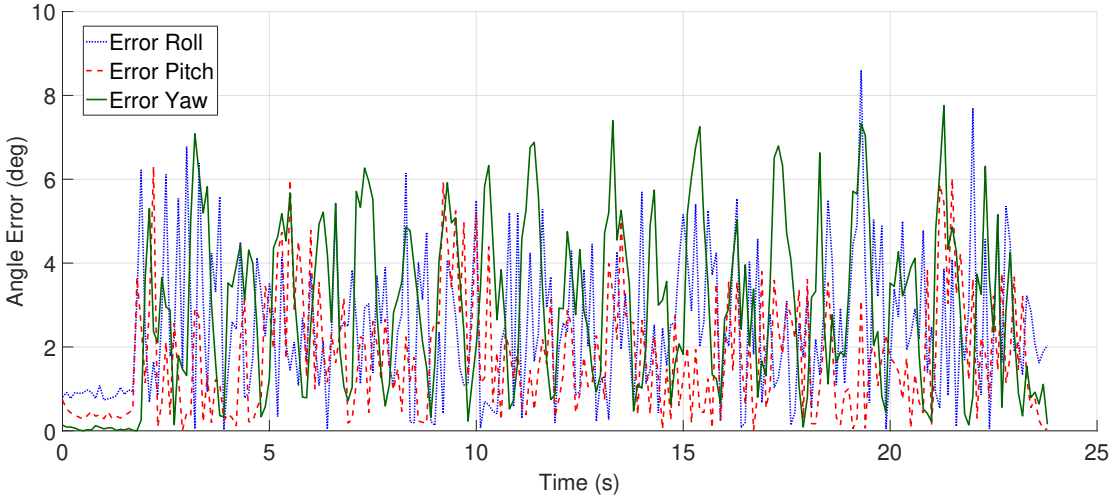


(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the rolling gait using the joint angles model.

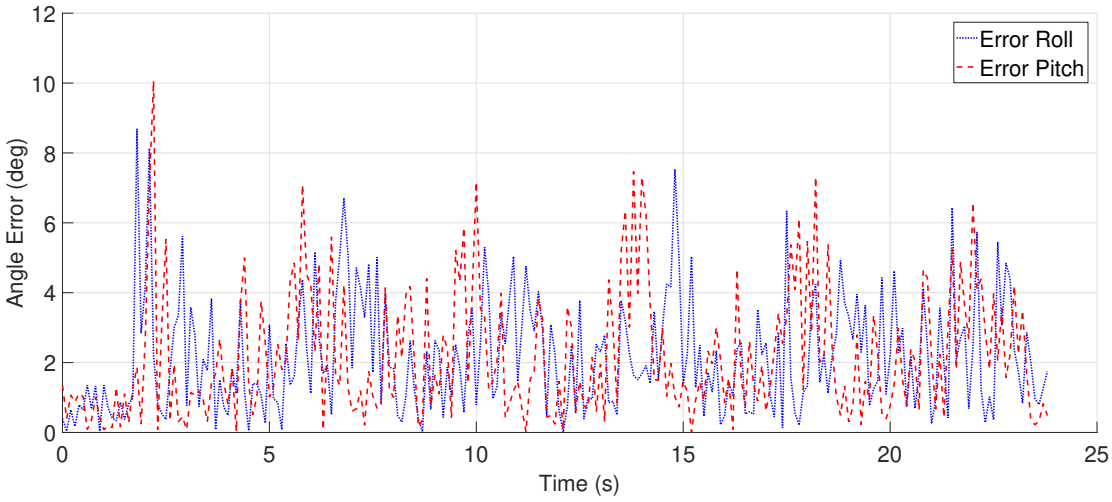


(b) Comparison between the predicted Euler angles and the sensor data for module 9 while performing the rolling gait using the joint angles model.

Figure D.13: Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the joint angles model.

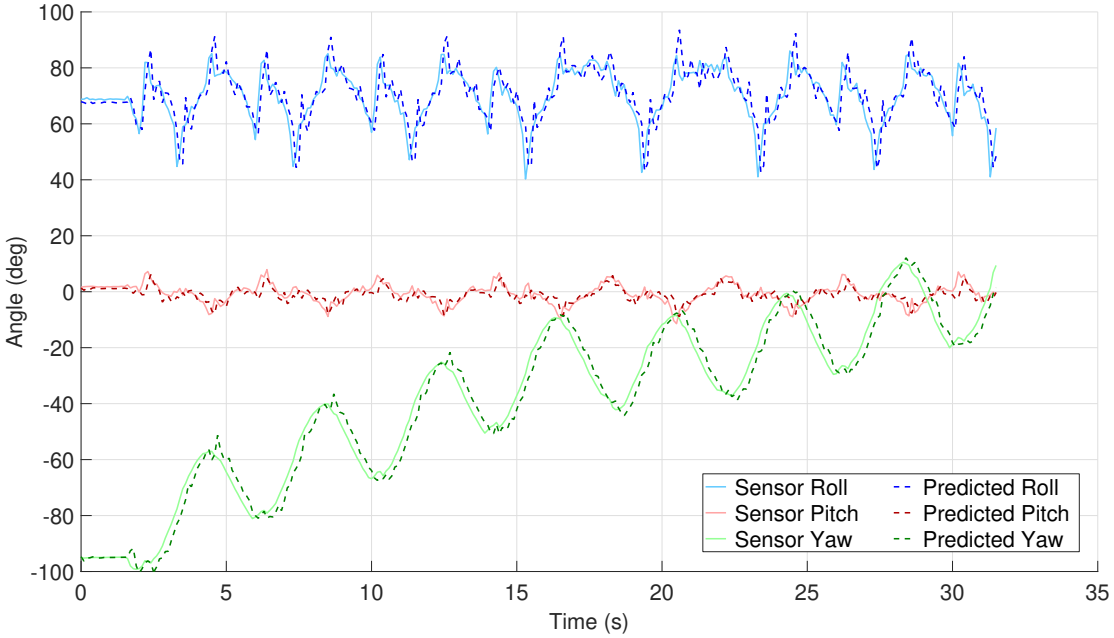


(a) Prediction error of module 0 using the rolling gait with the joint angles model.

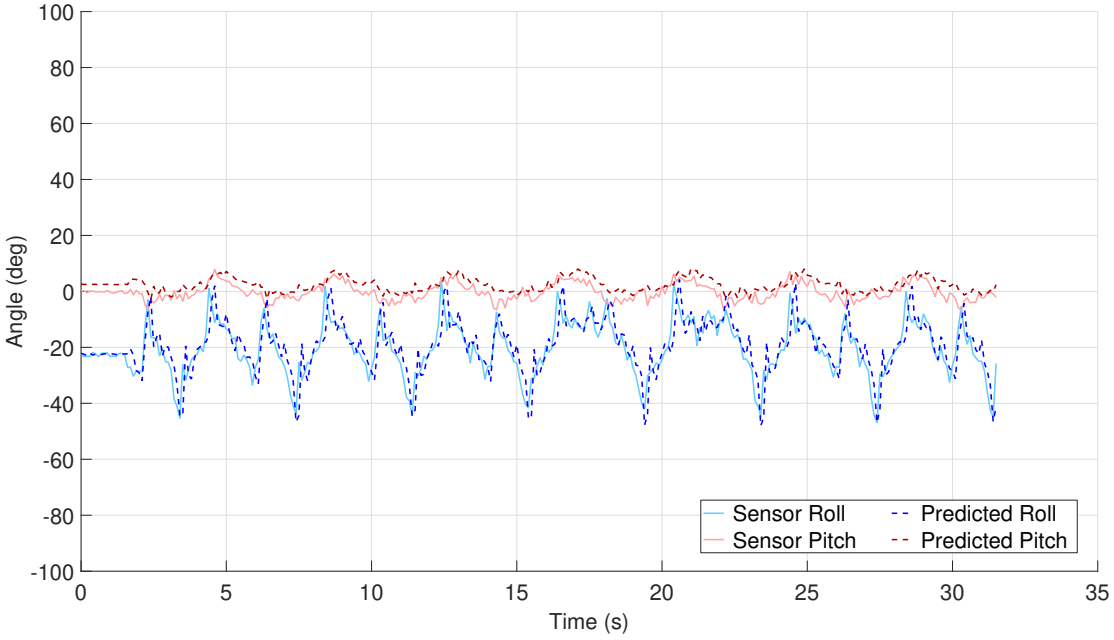


(b) Prediction error of module 9 using the rolling gait with the joint angles model.

Figure D.14: Predicted error of modules 0 and 9 with the SR-SSUKF while using the rolling gait with the joint angles model.

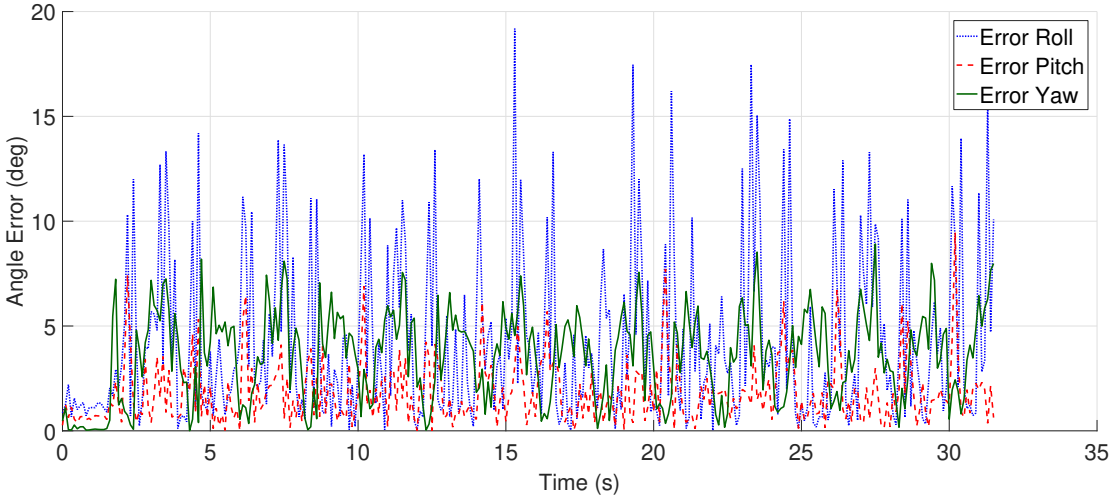


(a) Comparison between the predicted Euler angles and the sensor data for module 0 while performing the rotating gait using the joint angles model.

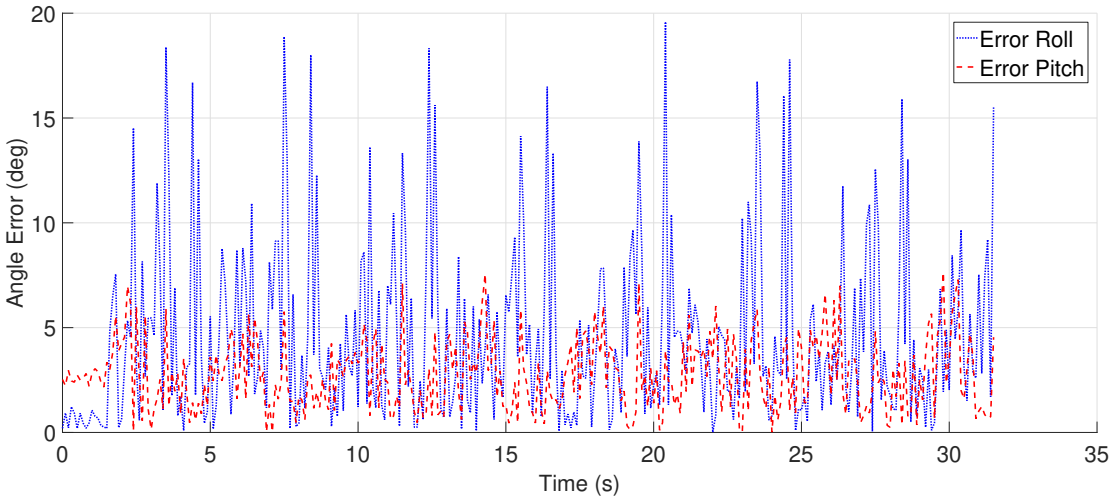


(b) Comparison between the predicted Euler angles and the sensor data for module 9 while performing the rotating gait using the joint angles model.

Figure D.15: Predicted pose of modules 0 and 9 with the SR-SSUKF while using the rotating gait with the joint angles model.



(a) Prediction error of module 0 using the rotating gait with the joint angles model.



(b) Prediction error of module 9 using the rotating gait with the joint angles model.

Figure D.16: Predicted error of modules 0 and 9 with the SR-SSUKF while using the rotating gait with the joint angles model.